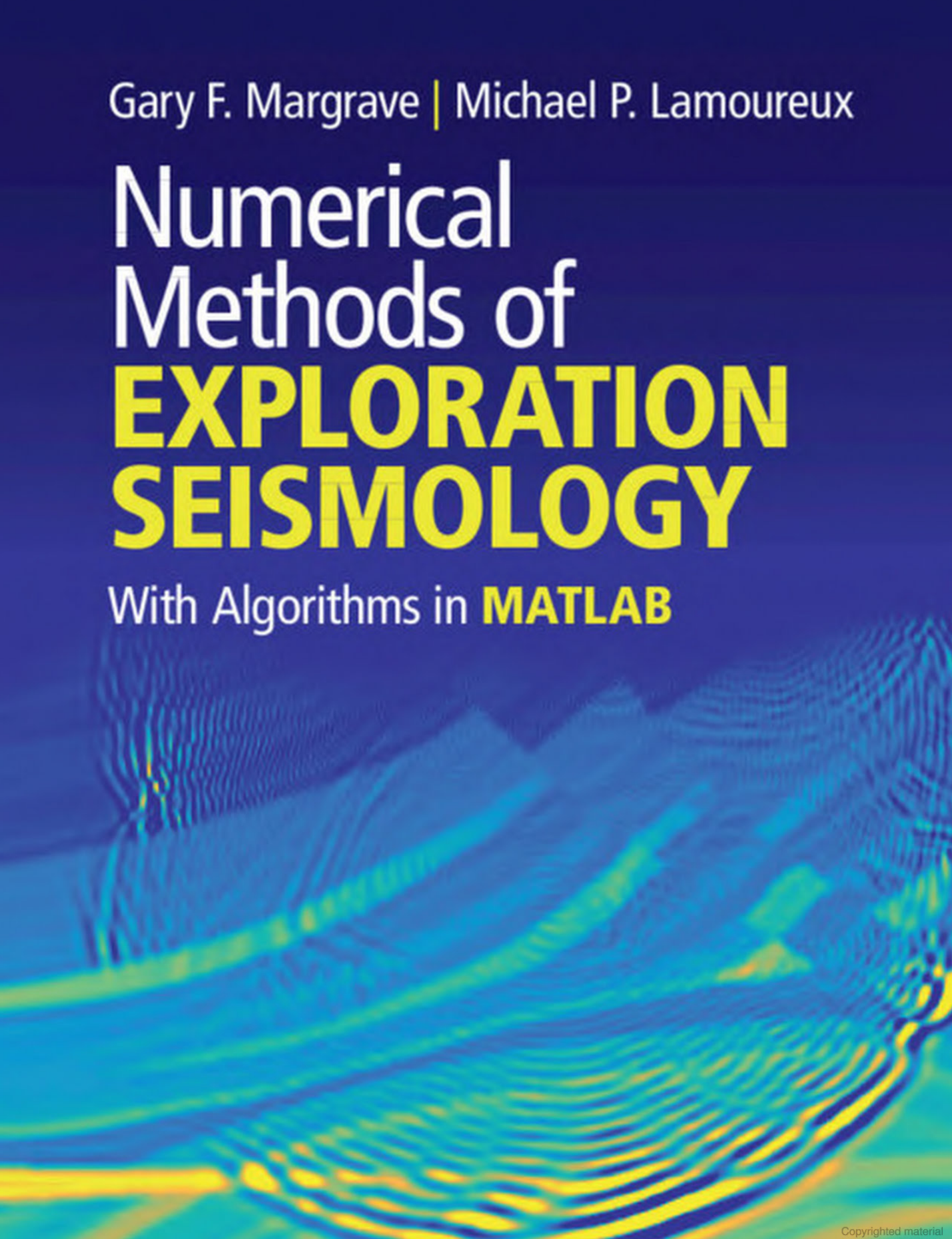
Gary F. Margrave | Michael P. Lamoureux

Numerical Methods of EXPLORATION SEISMOLOGY

With Algorithms in MATLAB



Numerical Methods of Exploration Seismology With Algorithms in MATLAB

Exploration seismology uses seismic imaging to form detailed images of the Earth's interior, enabling the location of likely petroleum targets. Owing to the size of seismic datasets, sophisticated numerical algorithms are required. This book provides a technical guide to the essential algorithms and computational aspects of data processing, covering the theory and methods of seismic imaging. The first part introduces an extensive online library of MATLAB seismic data-processing codes maintained by the CREWES project at the University of Calgary. Later chapters then focus on digital signal theory and relevant aspects of wave propagation and seismic modeling, followed by deconvolution and seismic migration methods. Presenting a rigorous explanation of how to construct seismic images, it provides readers with practical tools and codes to pursue research projects and analyses. It is ideal for advanced students and researchers in applied geophysics, and for practicing exploration geoscientists in the oil and gas industry.

Gary F. Margrave has extensive experience with seismic data in both the corporate and academic worlds. His career began with 15 years at Chevron, before 20 years as a professor of geophysics at the University of Calgary, where he taught courses on which this book is based. He then spent two years as Senior Geophysical Advisor at Devon Energy. He is now retired but still pursuing a vigorous research program.

Michael P. Lamoureux is a professor of mathematics at the University of Calgary, with a research focus on functional analysis and its application to physics, signal processing, and imaging. He has a keen interest in developing advanced mathematical methods for use in real industrial settings.

‘This book is a masterpiece in scope and content. It explains the essential algorithms and computational aspects of data processing, covering the theory and methods of seismic imaging. A particularly outstanding feature is that it gives useful methods and tools to pursue research projects and analyses – representing the way that things should be taught in the computer age. For this reason, it should be adopted in the undergraduate curriculum and will be a wonderful resource for graduate students and researchers in applied geophysics. Practicing geoscientists will also welcome this book as it will make their daily tasks easier and more productive.’

Enders Robinson, Professor Emeritus, Columbia University, New York City

‘The authors are to be commended for putting together this valuable resource which will instantly be highly useful to many geophysicists in the academic and industrial communities. The book is a pleasing and unusual mixture of rigorous geophysical signal processing theory and practical concepts, algorithms and code snippets. The MATLAB library functions and scripts that are provided or available for download will prove indispensable to all readers.’

Peter Cary, Chief Geophysicist, TGS Canada

Numerical Methods of Exploration Seismology

With Algorithms in MATLAB

GARY F. MARGRAVE
University of Calgary

MICHAEL P. LAMOUREUX
University of Calgary

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre,
New Delhi – 110025, India

79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107170148
DOI: 10.1017/9781316756041

First published 2019

Printed in the United Kingdom by TJ International Ltd, Padstow Cornwall

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging-in-Publication Data
Names: Margrave, Gary F., 1950– author. | Lamoureux, Michael P., author.
Title: Numerical methods of exploration seismology : with algorithms in MATLAB / Gary F. Margrave (University of Calgary), Michael P. Lamoureux (University of Calgary).
Description: Cambridge ; New York, NY : Cambridge University Press, 2019. | Includes bibliographical references and index.
Identifiers: LCCN 2018038868 | ISBN 9781107170148 (hardback)
Subjects: LCSH: Geological mapping–Data processing. | Three-dimensional imaging in geology. | Imaging systems in geology. | Earth (Planet)–Crust. | Geology, Structural–Data processing. | Seismic reflection method. | MATLAB.
Classification: LCC QE36.M33944 2019 | DDC 551.801/5186–dc23
LC record available at https://lccn.loc.gov/2018038868

ISBN 978-1-107-17014-8 Hardback

Additional resources for this publication at www.cambridge.org/nmes.

GFM: To Joan. Thank you so much for supporting me spiritually through the years. Your friendship, counsel, and encouragement have been my guiding lights.

MPL: To Olga and our children Gaby, Alex, Jakub, and Veronika. Thank you for your patience, support, and inspiration while writing this book, and more.

Contents

Preface

Exploration seismology is a complex technology that blends advanced physics, mathematics, and computation. Seismic imaging, an essential part of exploration seismology, has evolved over roughly 100 years of effort into a sophisticated imaging system capable of forming detailed 3D images of the interior of the Earth's crust. We have been involved in research, teaching, and practice in this field for many decades and this book is an outgrowth of our experience. With it we hope to bring a more detailed understanding of the methods of seismic imaging to a broad audience of scientists and engineers.

Often, the computational aspect is neglected in teaching because, traditionally, seismic processing software is part of an expensive and complex system. Also, understanding the numerical methods behind the software requires a considerable knowledge of digital signal theory, which is often omitted in a typical graduate curriculum. However, it is our opinion that true understanding only comes through mastering the computational aspects as well as the concepts and mathematics. We have often been surprised at the additional mental struggle required to transition from a formula in a book to an actual digital computation of the same formula. Even so, we have never regretted spending the extra time needed for that purpose.

This book is intended for those scientists who wish for an introduction to the computational aspects as well as the theory of seismic data processing. Such people may be graduate students at universities, professional data processors in seismic processing companies, researchers in energy companies, or literally anyone who wishes to gain a greater understanding of seismic data-processing algorithms. The appropriate background for this material is roughly that achieved at the B.Sc. level in physics, mathematics, or geoscience. Knowledge of vector calculus, undergraduate physics, some understanding of geophysics, and experience with a computer programming language (not necessarily MATLAB) are all assumed as background.

This book and the MATLAB library it describes are the product of many years of teaching and research at the University of Calgary and in industry. The first author began the development of the library while employed at Chevron and, with Chevron's permission, continued this development after joining the University of Calgary and the Consortium for Research in Elastic-Wave Exploration Seismology (CREWES) in 1995. He is now retired from the university, and still evolving the codes. The second author became involved because the mathematical complexity of the seismic imaging problem appealed to his expertise in functional analysis and signal processing, as a member of the university's Department of Mathematics and Statistics. Both authors are involved in an ongoing collaboration with one another and with other members of the CREWES project, and these codes are the fruit of that collaboration.

We have chosen to limit the scope of this volume primarily to those methods that can be regarded as "single-channel algorithms." This term means that these methods act one at a time on each of the millions or even billions of 1D time series that comprise a seismic dataset. Mostly in our final chapter, we do discuss some multichannel methods but these only scratch the surface of what can be found in our MATLAB library. Even to describe the single-channel methods in algorithmic detail requires a lengthy introduction to digital signal theory, our Chapters 2 and 3, and also a solid introduction to the relevant mathematical physics, Chapter 4, which then culminates in a detailed discussion of deconvolution methods in Chapter 5. Chapters 6 and 7 comprise an introduction to seismic migration methods.

We hope this book proves useful to our readers, and welcome any feedback. We realize that it will seem overly complex to some, while others will find it lacking in detail essential to their interests. We only ask for understanding that it is difficult to satisfy all readers, but we hope that all who make the required effort will find value here.

Online Resources

How to Obtain the MATLAB Codes

There are three types of MATLAB codes needed to fully utilize this book:

1. The standard MATLAB functions that come with a MATLAB installation. We recommend that you have standard MATLAB and the *signal* toolbox.
2. The library of MATLAB functions maintained and distributed by the CREWES project at the University of Calgary. To obtain this library, go to www.crewes.org/ResearchLinks/FreeSoftware/ and download *crewes.zip*. This is completely free for any purpose except resale. Also available there is a very early version of this text (from 2005).
3. The specific scripts that are presented in this text and used to make most of the figures. If you have already downloaded the CREWES library from the previous item, then you will already have these codes and they will appear in the subfolder *NMES_book*. Most of these codes are scripts that illustrate the use of many of the CREWES library tools. Nearly all of the figures in Chapters 1–5 have corresponding scripts in this subfolder that generate them. Chapters 6 and 7 also have support codes but not as extensively as the others. We have attempted to give the scripts suggestive names, but it usually takes some exploring to find exactly what you want. A useful MATLAB tool for this purpose is the multiple-file-search tool that is activated by typing ctrl-shift-f in the MATLAB command window. In the search for the code for a particular figure, you can sometimes succeed by using this search tool to find a particular text string that appears in the figure. Note also that each *code snippet* gives the file name where it is found, beneath the snippet. The corresponding plotting commands are usually found in a similarly named file.

The CREWES codes and the NMES book codes change regularly. It is a good idea to download a fresh copy of the archive at least monthly. Neither CREWES, the University of Calgary, nor the authors of this book make any guarantees of the correctness of these codes. We have tried our very best to produce mathematically correct and easy-to-understand codes but errors are almost certainly present. Use these codes at your own risk.

1 Introduction to MATLAB and Seismic Data

1.1 Scope and Prerequisites

This is a book about a complex and diverse subject: the numerical algorithms used to process exploration seismic data to produce images of the Earth's crust. The techniques involved range from simple and graphical to complex and numerical. More often than not, they tend toward the latter. The methods frequently use advanced concepts from physics, mathematics, numerical analysis, and computation. This requires the reader to have a background in these subjects at approximately the level of an advanced undergraduate or beginning graduate student in geophysics or physics. This need not include experience in exploration seismology, but such experience would be helpful.

Seismic datasets are often very large and have, historically, strained computer storage capacities. This, along with the complexity of the underlying physics, has also strongly challenged computation throughput. These difficulties have been a significant stimulus to the development of computing technology. In 1980, a 3D migration[1] was only possible in the advanced computing centers of the largest oil companies. At that time, a 50 000-trace 3D dataset would take weeks to migrate on a dedicated, multimillion-dollar computer system. Today, much larger datasets are routinely migrated by companies and individuals around the world, often on computers costing less than $5000. The effective use of this book, including working through the computer exercises, requires access to a significant machine (at least a late-model PC or Macintosh) with MATLAB installed and having significant memory and disk drive capacity.

Though numerical algorithms, coded in MATLAB, will be found throughout this book, this is not a book primarily about MATLAB. It is quite feasible for the reader to plan to learn MATLAB concurrently with working through this book, but a separate reference work on MATLAB is highly recommended. In addition to the reference works published by The MathWorks (the makers of MATLAB), there are many excellent independent guides in print such as Etter (1996), Hanselman and Littlefield (1998), Redfern and Campbell (1998), and the more recent Higham and Higham (2000). In addition, the student edition of MATLAB is a bargain and comes with a very good reference manual. If you already own a MATLAB reference, then stick with it until it proves inadequate. The website of The MathWorks is worth a visit because it contains an extensive database of books about MATLAB.

Though this book does not teach MATLAB at an introductory level, it illustrates a variety of advanced techniques designed to maximize the efficiency of working with large

[1] *Migration* refers to the fundamental step in creating an Earth image from scattered data.

datasets. As with many MATLAB applications, it helps greatly if the reader has had some experience with linear algebra. Hopefully, the concepts of matrices, row vectors, column vectors, and systems of linear equations will be familiar.

1.1.1 Why MATLAB?

A few remarks are appropriate concerning the choice of the MATLAB language as a vehicle for presenting numerical algorithms. Why not choose a more traditional language like C or Fortran, or an object-oriented language like C++ or Java?

MATLAB was not available until the latter part of the 1980s, and, prior to that, Fortran was the language of choice for scientific computations. Though C was also a possibility, its lack of a built-in facility for complex numbers was a considerable drawback. On the other hand, Fortran lacked some of C's advantages such as structures, pointers, and dynamic memory allocation.

The appearance of MATLAB changed the face of scientific computing for many practitioners, these authors included. MATLAB evolved from the Linpack package, which was familiar to Fortran programmers as a robust collection of tools for linear algebra. However, MATLAB also introduced a new vector-oriented programming language, an interactive environment, and built-in graphics. These features offered sufficient advantages that users found their productivity was significantly increased over the more traditional environments. Since then, MATLAB has evolved to have a large array of numerical tools, both commercial and shareware; excellent 2D and 3D graphics; object-oriented extensions; and a built-in interactive debugger.

Of course, C and Fortran have evolved as well. C has led to C++ and Fortran to Fortran90. Though both of these languages have their adherents, neither seems to offer as complete a package as does MATLAB. For example, the inclusion of a graphics facility in the language itself is a major boon. It means that MATLAB programs that use graphics are standard throughout the world and run the same on all supported platforms. It also leads to the ability to graphically display data arrays at a breakpoint in the debugger. These are useful practical advantages, especially when working with large datasets.

The vector syntax of MATLAB, once mastered, leads to more concise code than in most other languages. Setting one matrix equal to the transpose of another through a statement like `A=B';` is much more transparent than something like

```
do i=1,n
    do j=1,m
        A(i,j)=B(j,i)
    enddo
enddo
```

Also, for the beginner, it is actually easier to learn the vector approach, which does not require so many explicit loops. For someone well versed in Fortran, it can be difficult to unlearn this habit, but it is well worth the effort.

It is often argued that C and Fortran are more efficient than MATLAB and therefore more suitable for computationally intensive tasks. However, this view misses the big picture. What really matters is the efficiency of the entire scientific process, from the genesis of the

idea, through its rough implementation and testing, to its final polished form. Arguably, MATLAB is much more efficient for this entire process. The built-in graphics, interactive environment, large tool set, and strict run-time error checking lead to very rapid prototyping of new algorithms. Even in the more narrow view, well-written MATLAB code can approach the efficiency of C and Fortran. This is partly because of the vector language but also because most of MATLAB's number crunching actually happens in compiled library routines written in C.

Traditional languages like C and Fortran originated in an era when computers were room-sized behemoths and resources were scarce. As a result, these languages are oriented toward simplifying the task of the computer at the expense of the human programmer. Their cryptic syntax leads to efficiencies in memory allocation and computation speed that were essential at the time. However, times have changed and computers are relatively plentiful, powerful, and cheap. It now makes sense to shift more of the burden to the computer to free the human to work at a higher level. Spending an extra $100 to buy more RAM may be more sensible than developing a complex data-handling scheme to fit a problem into less space. In this sense, MATLAB is a higher-level language that frees the programmer from technical details to allow time to concentrate on the real problem.

Of course, there are always people who see these choices differently. Those in disagreement with the reasons cited here for MATLAB can perhaps take some comfort in the fact that MATLAB syntax is fairly similar to C or Fortran and translation is not difficult. Also, The MathWorks markets a MATLAB "compiler" that emits C code that can be run through a C compiler.

1.2 MATLAB Conventions Used in This Book

There are literally hundreds of MATLAB *functions* that accompany this book (and hundreds more that come with MATLAB). Since this is not a book about MATLAB, most of these functions will not be examined in any detail. However, all have full online documentation, and their code is liberally sprinkled with comments. It is hoped that this book will provide the foundation necessary to enable the user to use and understand all of these commands at whatever level necessary.

Typographic style variations are employed here to convey additional information about MATLAB functions. A function name presented like `plot` refers to a MATLAB function supplied by The MathWorks as part of the standard MATLAB package. A function name presented like ***`dbspec`*** refers to a function provided with this book. Moreover, the name *NMES Toolbox* refers to the entire collection of software provided in this book.

MATLAB code will be presented in small numbered packages entitled "code snippets." An example is the code required to convert an amplitude spectrum from linear to decibel scale (Code Snippet 1.2.1).

The actual MATLAB code is displayed in an upright typewriter font, while introductory remarks are *emphasized like this*. The code snippets do not employ typographic variations to indicate which functions are contained in the *NMES Toolbox* as is done in the text proper.

Code Snippet 1.2.1 This code computes a wavelet and its amplitude spectrum on both linear and decibel scales. It makes Figure 1.1. The final line prints the figure into an "eps" file, a detail that will not usually be included in subsequent code snippets.

```
1   [wavem,t]=wavemin(.001,20,1);
2   [Wavem,f]=fftrl(wavem,t);
3   Amp=abs(Wavem);
4   dbAmp=20*log10(Amp/max(Amp));
5   figure
6   subplot(3,1,1);plot(t,wavem);xlabel('time (sec)');xlim([0 .2])
7   subplot(3,1,2);plot(f,abs(Amp));xlabel('Hz');ylabel('linear scale');
8   ylim([0 1.01])
9   subplot(3,1,3);plot(f,dbAmp);xlabel('Hz');ylabel('db down')
10  prepfiga
11  bigfont(gcf,2,1)
12
13  print -deps .\intrographics\intro1a.eps
```

End Code

introcode / intro1 .m

It has proven impractical to discuss all of the input parameters for all of the programs shown in the code snippets. Those germane to the current topic are discussed, but the remainder are left for the reader to explore using MATLAB's interactive help facility.[2] For example, in Code Snippet 1.2.1, *wavemin* creates a minimum-phase wavelet sampled at 0.002 s, with a dominant frequency of 20 Hz and a length of 0.2 s. Then *fftrl* computes the Fourier spectrum of the wavelet and `abs` constructs the amplitude spectrum from the complex Fourier spectrum. Finally, the amplitude spectrum is converted to decibels (dB) on line 4.

A decibel scale is a logarithmic display of amplitude, which is extremely useful when the amplitude range extends over many orders of magnitude. Given an amplitude function like $A(f)$, where in this context we are speaking of the amplitude of a spectrum as a function of frequency, the decibel representation comes from the formula

$$A_{\mathrm{dB}}(f) = 20 \log_{10}\left(\frac{A(f)}{A_{\mathrm{ref}}}\right), \tag{1.1}$$

where we assume $A(f) > 0$ everywhere (if it is not obvious why, then you might want to review logarithms), and A_{ref} is a reference amplitude. Often the choice $A_{\mathrm{ref}} = \max(A(f))$ is made, which means that $A_{\mathrm{dB}}(f)$ will be all negative numbers except at the maximum, where it will be zero. Suppose $A(f) = A_{\mathrm{ref}}/2$; then $A_{\mathrm{dB}} = 20 \log_{10}(0.5) \approx -6$ dB. Using the properties of logarithms, if $A(f) = A_{\mathrm{ref}}/4$, then $A_{\mathrm{dB}} = 20 \log_{10}(\frac{1}{2}\frac{1}{2}) = 20[\log_{10}\frac{1}{2} + \log_{10}\frac{1}{2}] \approx -12$ dB. So, every -6 dB represents a decrease in amplitude by a factor of $\frac{1}{2}$. Similarly, if the reference amplitude is not the maximum, then every increase in amplitude

[2] To find out more about any of these functions and their inputs and outputs, type, for example, "help fftrl" at the MATLAB prompt.

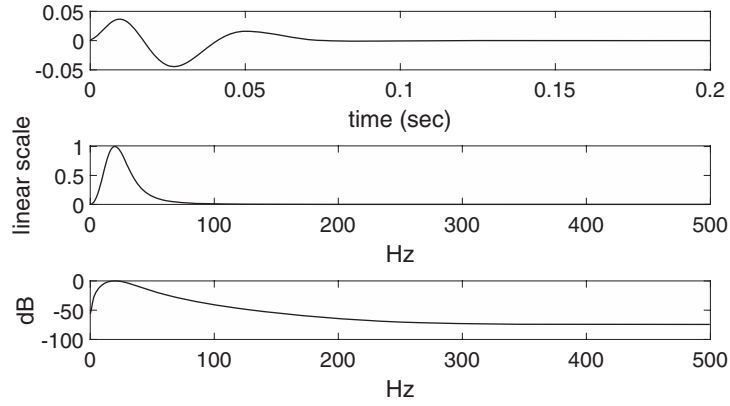


Figure 1.1 A minimum-phase wavelet (top), its amplitude spectrum plotted on a linear scale (middle), and its amplitude spectrum plotted on a decibel scale (bottom).

by a factor of 2 is a +6 dB increase. Sometimes we prefer to think in factors of 10, and it is similarly easy to show that -20 dB represents a decrease by a factor of $\frac{1}{10}$, while +20 dB is an increase by 10.

This example illustrates several additional conventions. Digital seismic traces[3] are discrete time series, but the common textbook convention of assuming a sample interval of unity in arbitrary units is not appropriate for practical problems. Instead, two vectors will be used to describe a trace, one to give its amplitude and the other to give the temporal coordinates for the first vector. Thus *wavemin* returns two vectors (that they are vectors is not obvious from the code snippet), with `wavem` being the wavelet amplitudes and `t` being the time coordinate vector for `wavem`. Thus the top graph of Figure 1.1 is created by simply cross-plotting these two vectors: `plot(t,wavem)`. Similarly, the Fourier spectrum, `Wavem`, has a frequency coordinate vector `f`. Temporal values must always be specified in seconds and will be returned in seconds, and frequency values must always be in hertz. (One hertz (Hz) is one cycle per second.) Milliseconds or radians/second specifications will never occur in code, though both cyclical frequency f and angular frequency ω may appear in formulas ($\omega = 2\pi f$).

Seismic traces will always be represented by column vectors, whether in the time or the Fourier domain. This allows an easy transition to 2D trace gathers, such as source records and stacked sections, where each trace occupies one column of a matrix. This column vector preference for signals can lead to a class of simple MATLAB errors for the unwary user. For example, suppose a seismic trace `s` is to be windowed to emphasize its behavior in one zone and de-emphasize it elsewhere. The simplest way to do this is to create a window vector *win* that is the same length as `s` and use MATLAB's .* (dot-star) operator to multiply

[3] A *seismic trace* is usually the recording of a single-component geophone or hydrophone. Less commonly, for a multicomponent geophone, it is the recording of one component.

Code Snippet 1.2.2 This code creates a synthetic seismogram using the *convolutional model* but then generates an error while trying to apply a boxcar window. (A boxcar window is a time series that is all zeros except for a selected range, where it is 1.0.)

```
1   [r,t]=reflec(1,.002,.2); %make a synthetic reflectivity
2   [w,tw]=wavemin(.002,20,.2); %make a wavelet
3   s=convm(r,w); % make a synthetic seismic trace
4   n2=round(length(s)/2);
5   win=0*(1:length(s));%initialize window to all zeros
6   win(n2-50:n2+50)=1;%100 samples in the center of win are 1.0
7   swin=s.*win; % apply the window to the trace
```

End Code

introcode / intro2 .m

each sample in the trace by the corresponding sample in the window. The temptation is to write a code something like Code Snippet 1.2.2.

MATLAB's response to this code is the error message

```
Error using  .*
Matrix dimensions must agree.
```

The error occurs because the code `1:length(s)` used on line 5 generates a 1×501 row vector like $1, 2, 3, \ldots, n$, where `n=length(s)`, while `s` is a 501×1 column vector. The .* operator requires both operands to have exactly the same geometry. The simplest fix is to write `swin=s.*win(:);`, which exploits the MATLAB feature that `a(:)` is reorganized into a column vector (see page 38 for a discussion) regardless of the actual size of `a`.

As mentioned previously, two-dimensional seismic data gathers will be stored in ordinary matrices. Each column is a single trace, and so each row is a *time slice*. A complete specification of such a gather requires both a time coordinate vector, `t`, and a space coordinate vector, `x`.

Rarely will the entire code from a function such as *`wavemin`* or *`reflec`* be presented. This is because the code listings of these functions can span many pages and contain much material that is not directly relevant or is outside the scope of this book. For example, there are often many lines of code that check input parameters and assign defaults. These tasks have nothing to do with numerical algorithms and so will not be presented or discussed. Of course, the reader is always free to examine the entire codes at leisure.

1.3 Seismic Wavelets

In the previous section, mention was made of a *minimum-phase wavelet*, of which there is a great variety, and there are also many other wavelet types. In seismology, a wavelet typically represents the waveform emitted by a seismic source and as possibly modified by data processing. All wavelet types have a specific mathematical form, which may have

variable parameters such as a dominant frequency, a wavelet length (in time), and perhaps a phase option. The mathematical formulas that these wavelets are based on will be presented later, in Section 4.7.1. For now, we will simply illustrate how to generate some basic wavelets and discuss their most obvious properties.

A minimum-phase wavelet (Figure 1.1) refers to a specific type of wavelet that is thought to be a good model for the waveform from an impulsive seismic source such as dynamite. For now, the reasons for the name "minimum phase" will not be investigated; rather, it is sufficient to realize that such wavelets are causal,[4] have a shaped amplitude spectrum with a dominant frequency, have an amplitude spectrum that must never vanish at any frequency,[5] and have a phase spectrum that is linked to the shape of the amplitude spectrum. (If these terms such as "spectrum," "amplitude," and "phase" are unfamiliar, rest assured that they will be explained fully in Chapter 2.) Minimum-phase wavelets are typically used to model raw or unprocessed data. While they arise naturally, they are not usually preferred for interpretation, because their peak energy is always delayed from $t = 0$. It is a major task of data processing to estimate the wavelet in the raw data (it is never known a priori) and design an operator to shape it to zero phase. Thus various zero-phase wavelets are typically used to represent processed seismic data.

Figures 1.2a and 1.2b show four typical wavelets in the time and frequency domains. The figures were created by Code Snippet 1.3.1. The minimum-phase wavelet, created by *wavemin*, shows the basic causal nature and the delay of the maximum energy from $t = 0$. The Ricker wavelet, created by *ricker*, is a popular wavelet for use in seismic interpretation. A frequent task is to create a synthetic seismogram from well information to compare with a final seismic image. The synthetic seismogram, s, is usually constructed from a convolutional model $s(t) = w(t) \bullet r(t)$, where r is the reflectivity function (derived from logging information), w is usually a zero-phase wavelet, very often a Ricker wavelet, and $\bullet$ represents convolution.[6] In Figure 1.2b, the amplitude spectra of these wavelets are shown, and the minimum-phase and Ricker wavelets are similar, with smoothly sloping spectra away from a dominant frequency. While the Ricker spectrum is only adjustable by shifting the dominant frequency, the decay rate of the minimum-phase wavelet's spectrum can be controlled with the parameter `m` on line 8. The other two wavelets, Ormsby (from *ormsby*) and Klauder (from *klauder*), have no single dominant frequency but instead have a broad, flat passband. The details of these wavelets and spectra can be controlled from the input parameters in the various programs. In Code Snippet 1.3.1, these parameters have been deliberately chosen to make the wavelets similar.

Comparing the different zero-phase wavelets, it is apparent that the Ricker wavelet has the least sidelobe energy but also has a greater width as measured between the first zero crossings on either side of the maximum. The choice of which zero-phase wavelet to use in creating a seismogram will depend upon the nature of the data and the data processing. The Klauder wavelet is intended for *correlated* Vibroseis data. Vibroseis data is usually

[4] A causal wavelet is one that vanishes before a specific time, usually taken to be $t = 0$, and persists arbitrarily long afterwards.

[5] Except possibly at a few isolated points.

[6] Convolution is described intuitively in Section 2.3.1.

Code Snippet 1.3.1 This code creates Figures 1.2a and 1.2b. Four wavelets are created: minimum phase with *wavemin*, Ricker with *ricker*, Ormsby with *ormsby*, and Klauder with *klauder*. The minimum-phase and Ricker wavelets have the same dominant frequency, specified on line 2. The Ormsby and Klauder wavelets have broad, flat passbands from `fmin` to `fmax`. The wavelets all have the same temporal length of `tlen`, which is set here to 2.0 s. This is much longer than usual and is done simply to force fine sampling in the frequency domain and make a better graph. The wavelets are created on lines 10–13 and their spectra are calculated on lines 15–18 using *fftrl*. Note the use of *linesgray* to plot, rather than *plot*. This makes figures with gray lines of different shades rather than color. The equivalent *plot* commands are shown commented out for comparison.

```
1   dt=.001;%time sample size
2   fdom=30;%dominant frequency for ricker and wavemin
3   fmin=10;%beginning of passband for ormsby and klauder
4   fmax=70;%end of passband for ormsby and klauder
5   tlen=2;%temporal length of each wavelet
6   slen=8;%sweep length (klauder)
7   staper=.5;%sweep taper (klauder)
8   m=3.5;%spectral decay control in wavemin
9
10  [wm,twm]=wavemin(dt,fdom,tlen,m);
11  [wr,twr]=ricker(dt,fdom,tlen);
12  [wo,two]=ormsby(fmin-5,fmin,fmax,fmax+10,tlen,dt);
13  [wk,twk]=klauder(fmin,fmax,dt,slen,tlen,staper);
14
15  [Wm,fwm]=fftrl(wm,twm);
16  [Wr,fwr]=fftrl(wr,twr);
17  [Wo,fwo]=fftrl(wo,two);
18  [Wk,fwk]=fftrl(wk,twk);
19
20  figure
21  inc=.1;
22  %plot(twm,wm,twr,wr+inc,two,wo+2*inc,twk,wk+3*inc);
23  linesgray({twm,wm,'-',.4,0},{twr,wr+inc,'-',.9,.3},...
24      {two,wo+2*inc,'-',.9,.5},{twk,wk+3*inc,'-',1.1,.7});
25  xlabel('time (sec)')
26  xlim([-.25 .25]);ylim([-.1 .45])
27  legend('Minimum phase','Ricker','Ormsby','Klauder')
28  grid
29  prepfig;bigfont(gca,1.8,1)
30
31  figure
32  %plot(fwm,abs(Wm),fwr,abs(Wr),fwo,abs(Wo),fwk,abs(Wk))
33  linesgray({fwm,abs(Wm),'-',.4,0},{fwr,abs(Wr),'-',.9,.3},...
34      {fwo,abs(Wo),'-',.9,.5},{fwk,abs(Wk),'-',1.1,.7});
35  xlabel('frequency (Hz)')
36  xtick([0 10 30 50 70 100 150])
37  legend('Minimum phase','Ricker','Ormsby','Klauder')
38  xlim([0 150]);ylim([0 1.1])
39  grid
40  prepfig;bigfont(gca,1.8,1)
41
```

End Code

introcode / waveletsfig .m

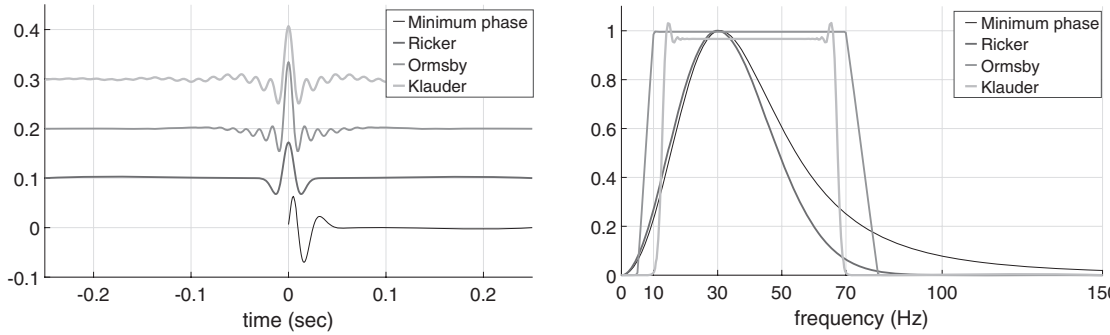


Figure 1.2a (left) Four common seismic wavelets. The minimum-phase wavelet is a causal wavelet, which is a model for an impulsive source like dynamite. The Ricker wavelet is zero phase with minimal sidelobes. The Ormsby wavelet is also zero phase but with much larger sidelobes. The zero-phase Klauder wavelet is a common model for Vibroseis data. The wavelets are plotted with vertical shifts to avoid plotting on top of each other.

Figure 1.2b (right) The amplitude spectra of the wavelets of Figure 1.2a. The spectra of the minimum-phase and Ricker wavelets decay smoothly away from a dominant frequency. The Ormsby and Klauder wavelets have a flat, broad passband and then decay sharply at the limits of the passband. The ears at the edges of the Klauder passband are related to the length of the sweep taper.

acquired with a source signature known as a sweep. A sweep is a temporally long[7] signal in which the vibrator runs through a prescribed range of frequencies called the *swept band*. When equal power is given to each frequency, the sweep is known as a linear sweep, and this is the usual case. Such data can be modeled as $s_{\mathrm{v}}(t) = \sigma(t) \bullet I(t)$, where $\sigma(t)$ is the sweep and $I(t)$ is the Earth's impulse response that would be recorded from a perfectly impulsive source. After acquisition, the raw data is crosscorrelated with the sweep. We can represent the correlated data as $s_{\mathrm{c}} = \sigma(t) \otimes s_{\mathrm{v}}(t) = w_{\mathrm{k}}(t) \bullet I(t)$, where $\otimes$ means crosscorrelation and w_{k} is the Klauder wavelet defined by $w_{\mathrm{k}}(t) = \sigma(t) \otimes \sigma(t)$. Thus the Klauder wavelet is the crosscorrelation of the sweep with itself, or the *autocorrelation* of the sweep. There is a theoretical formula for the Klauder wavelet, but it is more realistic to actually construct the sweep and numerically autocorrelate it, and this is what `klauder` does. In Figure 1.2b, the Klauder spectrum is noticeably more narrow than the Ormsby one. This is because, while the sweep begins and ends at 10 and 70 Hz, respectively, its amplitude must rise gradually from zero to full strength because it is used to drive a large machine. The length of time over which this happens is called the sweep taper and is applied at both ends of the sweep. The taper causes reduced power at the beginning and end of the sweep and is also responsible for the "ears" seen at about 15 and 65 Hz. Reducing the length of the taper will broaden the spectrum but increase the height of the ears.

The Ormsby wavelet is often used to represent seismic data that has been deliberately processed to have a flat passband over some prescribed range. This produces a very narrow central wavelet peak but with quite a lot of sidelobe energy. The specification of an Ormsby

[7] Sweep lengths ranging from 4 s to as long as 20 s are common in practice.

wavelet is usually done by giving the four *Ormsby parameters*, which are called f_1, f_2, f_3, f_4, with the following meanings:

- f_1: high end of the low-frequency stop band. There is no signal for $f < f_1$.
- f_2: low end of the passband. The amplitude ramps up linearly for $f_1 \leq f \leq f_2$.
- f_3: high end of the passband. The amplitude is constant for $f_2 \leq f \leq f_3$.
- f_4: low end of the high-frequency stop band. There is no signal for $f > f_4$.

The theoretical Ormsby wavelet is infinitely long, and any practical version must be truncated. This truncation causes the stop bands to have some nonzero energy.

Each of these wavelets is normalized such that a sine wave of the dominant frequency is unchanged in amplitude when convolved with the wavelet.

1.4 Dynamic Range and Seismic Data Display

Seismic data tends to be a challenge to computer graphics as well as to computer capacity. A single seismic record can have a tremendous amplitude range. In speaking of this, the term *dynamic range* is used, which refers to a span of real numbers. The range of numerical voltages that a seismic recording system can faithfully handle is called its dynamic range. For example, current digital recording systems use a fixed instrument gain and represent amplitudes as 24-bit integer computer words.[8] The first bit is used to record the sign, while the last bit tends to fluctuate randomly, so effectively 22 bits are available. This means that the amplitude range can be as large as $2^{22} \approx 10^{6.6}$. Using the definition of a decibel given in Eq. (1.1), this corresponds to about 132 dB, an enormous spread of possible values. This 132 dB range is never usually fully realized when recording signals, for a variety of reasons, the most important being the ambient noise levels at the recording site and the instrument gain settings.

In a fixed-gain system, the instrument gain settings are determined to minimize *clipping* by the analog-to-digital converter while still preserving the smallest possible signal amplitudes. This usually means that some clipping will occur on events nearest the seismic source. A very strong signal should saturate 22–23 bits, while a weak signal may affect only the lowest several bits. Thus the *precision*, which refers to the number of significant digits used to represent a floating-point number, steadily declines from the largest to the smallest number in the dynamic range.

1.4.1 Single-Trace Plotting and Dynamic Range

Figure 1.3 was produced with Code Snippet 1.4.1 and shows two real seismic traces recorded in 1997 by CREWES.[9] This type of plot is called a *wiggle trace display*.

[8] Previous systems used a 16-bit word and variable gain. A four-bit gain word, an 11-bit mantissa, and a sign bit determined the recorded value.

[9] CREWES is an acronym for the *Consortium for Research in Elastic Wave Exploration Seismology* at the University of Calgary.

Code Snippet 1.4.1 This code loads near and far offset test traces, computes the *Hilbert envelopes* of the traces (with a decibel scale), and produces Figures 1.3 and 1.4.

```
clear; load testtrace.mat
figure
subplot(2,1,1);plot(t,tracefar,'k');
title('1000 m offset');xlabel('seconds')
subplot(2,1,2);plot(t,tracenear,'k');
title('10 m offset');xlabel('seconds')
prepfig
bigfont(gcf,1.3,1)

print -depsc intrographics\intro3.eps

envfar = abs(hilbert(tracefar)); %compute Hilbert envelope
envnear = abs(hilbert(tracenear)); %compute Hilbert envelope
envdbfar=todb(envfar,max(envnear)); %decibel conversion
envdbnear=todb(envnear); %decibel conversion
figure
plot(t,[envdbfar envdbnear],'k');xlabel('seconds');ylabel('decibels');
grid;axis([0 3 -140 0])
prepfig
bigfont(gcf,1.3,1)
ht=text(1.2,-40,'Near trace envelope');
fs=get(ht,'fontsize');
set(ht,'fontsize',2*fs);
ht=text(1,-100,'Far trace envelope');
set(ht,'fontsize',2*fs);

print -depsc intrographics\intro3_1.eps
```

End Code

introcode / intro3 .m

The upper trace, called `tracefar`, was recorded on the vertical component of a three-component geophone placed about 1000 m from the surface location of a dynamite shot.[10] The lower trace, called `tracenear`, was similar except that it was recorded only 10 m from the shot. (Both traces come from the shot record shown in Figures 1.10a and 1.10b.) The dynamite explosion was produced with 4 kg of explosive placed at 18 m depth. Such an explosive charge is about 2 m long, so the distance from the top of the charge to the closest geophone was about $\sqrt{16^2 + 10^2} \approx 19$ m, while that to the farthest geophone was about $\sqrt{16^2 + 1000^2} \approx 1000$ m. The vertical axes for the two traces indicate the very large amplitude difference between them. If they were plotted on the same axes, `tracefar` would appear as a flat, horizontal line next to tracenear.

[10] A dynamite charge is usually placed below ground in a borehole drilled by a special shot-hole drilling machine. Such boreholes are typically 2 inches in diameter and 3–20 m deep. Larger charges are placed in deeper holes. A 1 kg charge size placed 10 m deep is common and is packaged as a cylinder about 1 ft long and 2 inches in diameter. The surface location of a dynamite shot refers to the location of the top of the shot hole at the Earth's surface.

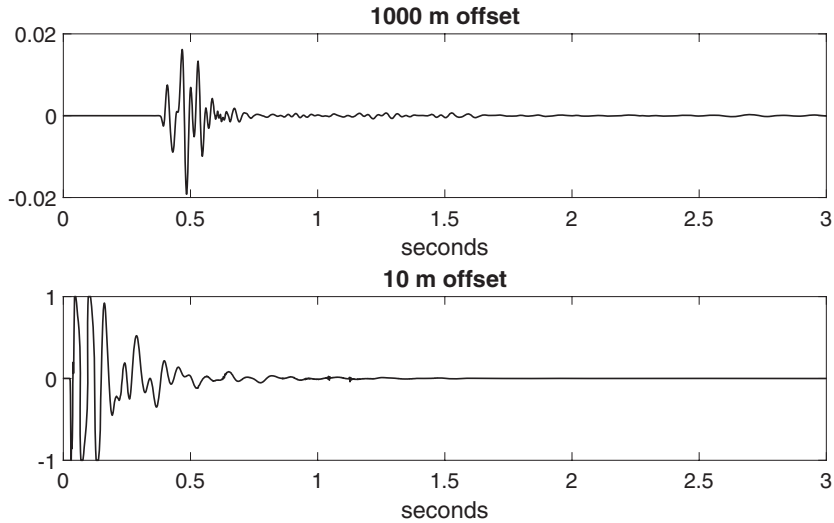


Figure 1.3 Top: A real seismic trace recorded about 1000 m from a dynamite shot. Bottom: A similar trace recorded only 10 m from the same shot. See Code Snippet 1.4.1.

Figure 1.4 (also produced with Code Snippet 1.4.1) shows a more definitive comparison of the amplitudes of the two traces. Here the *trace envelopes* are compared using a decibel scale. A trace envelope is a mathematical estimate of a bounding curve for the signal (this will be investigated more fully later, in Section 2.4.6) and is computed using `hilbert`, which computes the complex, analytic trace (Taner et al., 1979; Cohen, 1995) and then takes the absolute value. The conversion to decibels is done with the convenience function `todb` (for which there is an inverse, `fromdb`). The function `todb` implements Eq. (1.1) for both real and complex signals. (In the latter case, `todb` returns a complex signal whose real part is the amplitude in decibels and whose imaginary part is the phase.) By default, the maximum amplitude for the decibel scale is the maximum absolute value of the signal, but this may also be specified as the second input to `todb`. The function `fromdb` reconstructs the original signal given the output of `todb`.

The close proximity of `tracenear` to a large explosion produces a very strong first arrival, while later information (at 3 s) has decayed by ∼72 dB. (To gain familiarity with decibel scales, it is useful to note that 6 dB corresponds to a factor of 2. Thus 72 dB represents about 72/6 ∼ 12 doublings, or a factor of $2^{12} = 4096$.) Alternatively, `tracefar` shows peak amplitudes that are 40 dB (a factor of $2^{6.7} \sim 100$) weaker than `tracenear`.

The *first break time* is the best estimate of the arrival time of the first seismic energy. For `tracefar`, this is about 0.380 s, while for `tracenear` it is about 0.02 s. On each trace, energy before this time cannot have originated from the source detonation and is usually taken as an indication of ambient noise conditions. That is, it is due to seismic noise caused by wind, traffic, and other effects outside the seismic experiment. Only for `tracefar` is the first arrival late enough to allow a reasonable sampling of the ambient noise conditions. In this case, the average background noise level is about 120 to 130 dB below the peak signals

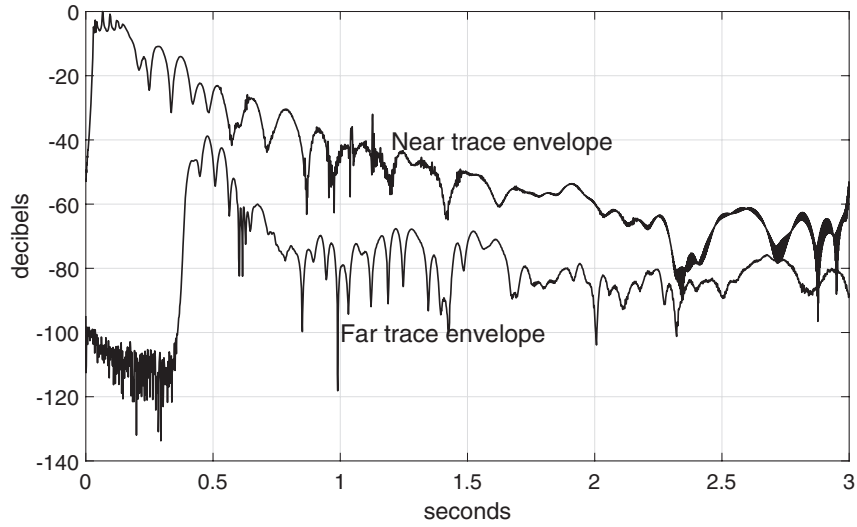


Figure 1.4 The envelopes of the two traces in Figure 1.3 plotted on a decibel scale. The far offset trace is about 40 dB weaker than the near offset one, and the total dynamic range is about 120 dB. See Code Snippet 1.4.1.

on `tracenear`. This is very near the expected instrument performance. It is interesting to note that the largest peaks on `tracenear` appear to have square tops, indicating clipping, at an amplitude level of 1.0. This occurs because the gain settings of the recording system were set to just clip the strongest arrivals and therefore distribute the available dynamic range over an amplitude band just beneath these strong arrivals.

Dynamic range is an issue in seismic display as well as in recording. It is apparent from Figure 1.3 that both signals fade below the visual threshold at about 1.6 s. Checking with the envelopes in Figure 1.4, this suggests that the dynamic range of this display is about 40–50 dB. This limitation is controlled by two factors: the total width allotted for the trace plot and the minimum perceivable trace deflection. In Figure 1.3, this width is about 1 inch and the minimum discernible wiggle is about 0.01 inches. Thus the dynamic range is about $10^{-2} \sim 40$ dB, in agreement with the earlier visual assessment.

It is very important to realize that seismic displays have limited dynamic range. This means that what you see is not always what you've got. For example, if a particular seismic display being interpreted for exploration has a dynamic range of, say, 20 dB, then any spectral components (i.e., frequencies in the Fourier spectrum) that are more than 20 dB down will not affect the display. If these weak spectral components are signal rather than noise, then the display does not allow the optimal use of the data. This is an especially important concern for wiggle trace displays of multichannel data, where each trace gets about one-tenth of an inch of display space.

More popular than the wiggle trace display is the wiggle-trace, variable-area (WTVA) display. The function *wtva* (Code Snippet 1.4.2) was used to create Figure 1.5, where the two display types are contrasted using `tracefar`. The WTVA display fills in the *peaks* of the seismic trace (or *troughs* if the polarity is reversed) with solid color. Doing this

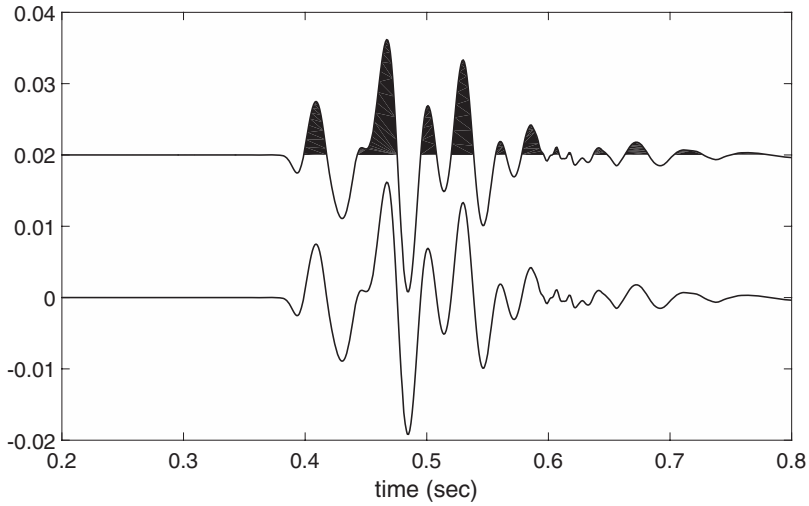


Figure 1.5 A portion of the seismic trace in Figure 1.3 is plotted in WTVA format (top) and wiggle trace format (bottom). See Code Snippet 1.4.2.

Code Snippet 1.4.2 The same trace is plotted with *wtva* and `plot`. Figure 1.5 is the result.

```
clear;load testtrace.mat
figure
plot(t,tracefar,'k')
[h,hva]=wtva(tracefar+.02,t,'k',.02,1,-1,1);
axis([.2 .8 -.02 .04])
xlabel('time (sec)')
prepfiga
bigfont(gcf,1.7,1)

print -deps .\intrographics\intro3a
```

End Code

introcode / intro3a .m

requires determining the *zero crossings* of the trace, which can be expensive if precision is necessary. The function *wtva* just picks the sample closest to each zero crossing. For more precision, the final argument of *wtva* is a resampling factor, which causes the trace to be resampled and then plotted. Also, *wtva* works like MATLAB's low-level function `line` in that it does not clear the figure before plotting. This allows *wtva* to be called repeatedly in a loop to plot a seismic section. The return values of *wtva* are MATLAB graphics handles for the "wiggle" and the "variable area" that can be used to further manipulate their graphic properties. (For more information, consult your MATLAB reference.)

Clipping was mentioned previously in conjunction with recording but also plays a role in display. Clipping refers to the process of setting all values on a trace that are greater than a clip level equal to that level. This can have the effect of moving the dynamic range

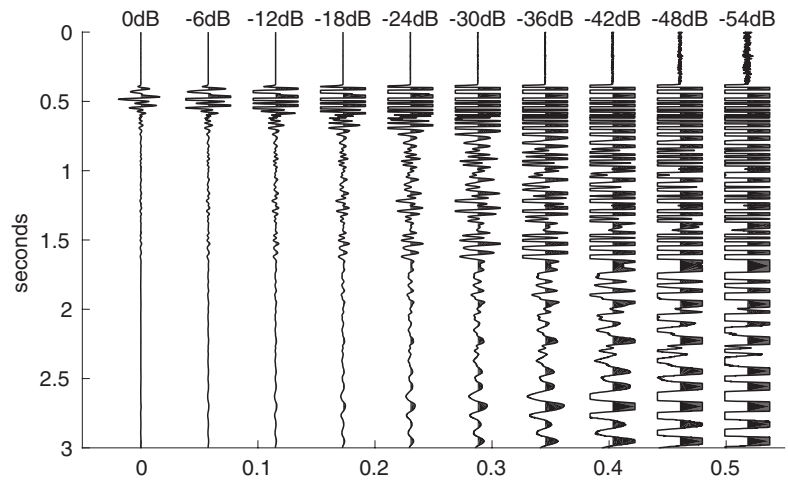


Figure 1.6 The seismic trace in Figure 1.3 is plotted repeatedly with different clip levels. The clip levels are annotated on each trace. See Code Snippet 1.4.3.

of a plot display into lower amplitude levels. Figure 1.6 results from Code Snippet 1.4.3 and shows the effect of plotting the same trace (`tracefar`) at progressively higher clip levels. The function *clip* produces a clipped trace, which is subsequently rescaled so that the clip level has the same numerical value as the maximum absolute value of the original trace. The effect of clipping is to make the weaker amplitudes more evident at the price of severely distorting the stronger amplitudes. Clipping does not increase the dynamic range; it just shifts the available range to a different amplitude band.

Exercises

1.4.1 Use MATLAB to load the test traces shown in Figure 1.3 and display them. By appropriately zooming your plot, estimate the first break times as accurately as you can. What is the approximate velocity of the material between the shot and the geophone? (You may find the function *simplezoom* useful. After creating your graph, type `simplezoom` at the MATLAB prompt and then use the *left* mouse button to draw a zoom box. A double-click will unzoom.)

1.4.2 Use MATLAB to load the test traces shown in Figure 1.3 and compute the envelopes as shown in Code Snippet 1.4.1. For either trace, plot both the wiggle trace and its envelope and show that the trace is contained within the bounds defined by $\pm$ envelope.

1.4.3 What is the dynamic range of a seismic wiggle trace display plotted at 10 traces/inch? What about 30 traces/inch?

Code Snippet 1.4.3 This code makes Figure 1.6. The test trace is plotted repeatedly with progressively higher clip levels. Line 5 defines the clip level, line 6 clips the trace by calling *clip*, and line 7 rescales the clipped trace to have the same maximum amplitude as the original trace. The last line creates an eps (Encapsulated Postscript) file from the figure.

```
1   clear;load testtrace.mat
2   figure
3   amax=max(abs(tracefar));
4   for k=1:10
5      clip_level= amax*(.5)^(k-1);
6      trace_clipped=clip(tracefar,clip_level);
7      trace_adj=trace_clipped*amax/max(abs(trace_clipped));
8      wtva(trace_adj+(k-1)*3*amax,t,'k',(k-1)*3*amax,1,1,1);
9      ht=text((k-1)*3*amax,-.05,[int2str(-(k-1)*6) 'db']);
10     set(ht,'horizontalalignment','center')
11  end
12  flipy;ylabel('seconds');
13  prepfig
14  bigfont(gcf,1.7,1)
15  xlim([-.05 .55])
16
17  print -depsc .\intrographics\intro3b
```

End Code

introcode / intro3b .m

1.4.2 Multichannel Seismic Display

Figure 1.6 illustrates the basic idea of a multichannel WTVA display. Assuming *ntr* traces, the plot width is divided into *ntr* equal segments to display each trace. A trace is plotted in a given plot segment by adding an appropriate constant to its amplitude. In the general case, these segments may overlap, allowing traces to overplot one another. After a clip level is chosen, the traces are plotted such that an amplitude equal to the clip level gets a *trace excursion* to the edges of the trace plot segment. For hard-copy displays, the traces are usually plotted at a specified number per inch.

Figure 1.7a is a synthetic seismic section made by creating a single synthetic seismic trace and then replicating it along a sinusoidal (with respect to x) trajectory. Figure 1.7b is a zoomed portion of the same synthetic seismic section. The function *plotseis* made the plot (see Code Snippet 1.4.3), and clipping was intentionally introduced. The square troughs in several events (e.g., near 0.4 s) are the signature of clipping. The function *plotseis* provides facilities to control clipping, produce either wiggle trace or WTVA displays, change polarity, and more. By default, *plotseis* plots on the current axis, so here we precede it with the *figure* command to open up a new window. The subsequent *bigfont* command makes the axis font bold for improved readability. By default, *plotseis* puts the horizontal axis on top, but this can be shifted to the bottom with the command `set(gca,'xaxislocation','bottom')`.

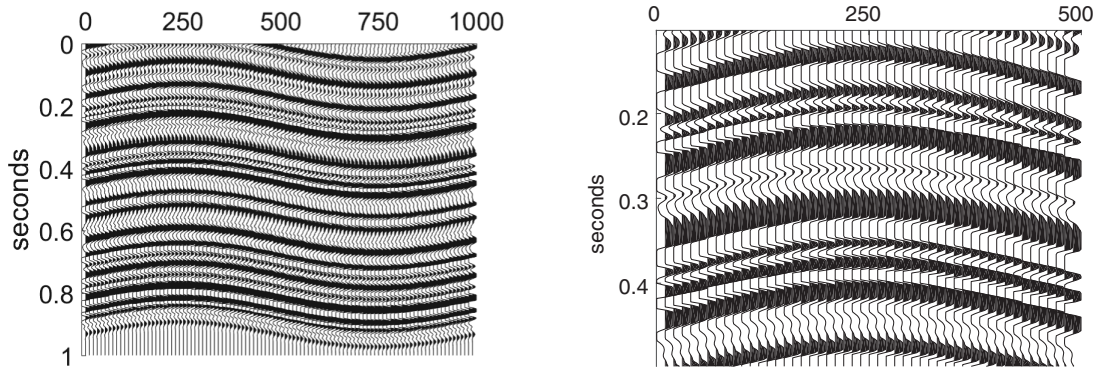


Figure 1.7a (left) A synthetic seismic section plotted in WTVA mode with clipping. See Code Snippet 1.4.4.

Figure 1.7b (right) A zoom (enlargement of a portion) of Figure 1.7a. Note the clipping indicated by square-bottomed troughs.

Code Snippet 1.4.4 Here we create a simple synthetic seismic section and plot it as a WTVA plot with clipping. Figure 1.7a is the result.

```
1   global NOSIG;NOSIG=1;
2   [r,t]=reflec(1,.002,.2,3,sqrt(pi));%make reflectivity
3   nt=length(t);
4   [w,tw]=wavemin(.002,20,.2);%make wavelet
5   s=convm(r,w);%make convolutional seismogram
6   ntr=100;%number of traces
7   seis=zeros(length(s),ntr);%preallocate seismic matrix
8   shift=round(20*(sin((1:ntr)*2*pi/ntr)+1))+1; %a time shift
9   %load the seismic matrix
10  for k=1:ntr
11     seis(1:nt-shift(k)+1,k)=s(shift(k):nt);
12  end
13  x=(0:99)*10; %make an x coordinate vector
14  figure
15  plotseis(seis,t,x,1,5,1,1,'k');ylabel('seconds')
16  xtick(0:250:1000)
17  bigfont(gcf,2,1)
18
19  print -depsc .\intrographics\intro4
```

End Code

introcode / intro4 .m

Code Snippet 1.4.4 illustrates the use of global variables to control plotting behavior. The global variable `NOSIG` controls the appearance of a *signature* at the bottom of a figure created by *`plotseis`*. If `NOSIG` is set to zero, then *`plotseis`* will annotate the date, time, and user's name in the bottom right corner of the plot. This is useful in classroom settings when many people are sending nearly identical displays to a shared printer. The user's name is defined as the value of another global variable, `NAME_` (the capitalization and the

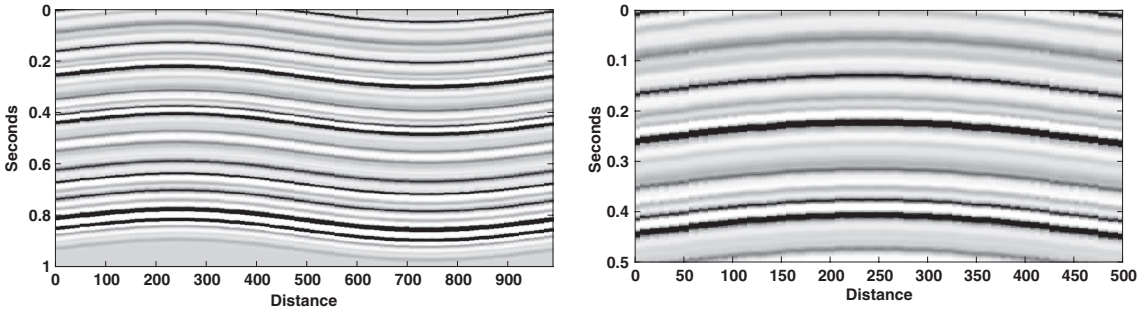


Figure 1.8a (left) A synthetic seismic section plotted as an image using `plotimage`. Compare with Figure 1.7a.
Figure 1.8b (right) A zoom (enlargement of a portion) of Figure 1.8a. Compare with Figure 1.7b.

underscore are important). Display control through global variables is also used in other utilities in the *NMES Toolbox* (see page 19). An easy way to ensure that these variables are always set as you prefer is to include their definitions in your *startup.m* file (see your MATLAB reference for further information).

The popularity of the WTVA display has resulted partly because it allows an assessment of the seismic waveform (if unclipped) as it varies along an event of interest. However, because its dynamic range varies inversely with trace spacing, it is less suitable for small displays of densely spaced data such as those on a computer screen. It also falls short for display of multichannel Fourier spectra, where the wiggle shape is not usually desired. For these purposes, *image* displays are more suitable. In this technique, the display area is divided into small rectangles (pixels) and these rectangles are assigned a color (or gray level) according to the amplitude of the samples within them. On the computer screen, if the number of samples is greater than the number of available pixels, then each pixel represents the average of many samples. Conversely, if the number of pixels exceeds the number of samples, then a single sample can determine the color of many pixels, leading to a blocky appearance.

Figures 1.8a and 1.8b display the same data as Figures 1.7a and 1.7b but use the function `plotimage` in place of `plotseis` in Code Snippet 1.4.4. The syntax to invoke `plotimage` is `plotimage(seis,t,x)`. This may also be called as `plotimage(seis)`, in which case it creates x and t coordinate vectors as simply column and row numbers. Unlike `plotseis`, `plotimage` adorns its figure window with controls to allow the user to interactively change the polarity, brightness, and color map, and determine the data-scaling scheme. Although controls appear by default when `plotimage` is used, they have been suppressed in these figures because they do not display well in print. The controls allow interactive setting of the clip level and other display properties. To suppress these controls, right-click in the primary axes and select "Hide Image Controls." To reverse this, simply right-click again in the primary axes and select "Show Image Controls." There is a version of `plotseis` called `plotseismic` which includes graphical image controls.

By default, `plotimage` uses a gray-level color map, defined by `seisclrs`, which is quite nonlinear. In a typical linear map, 64 separate gray levels are defined and a linear

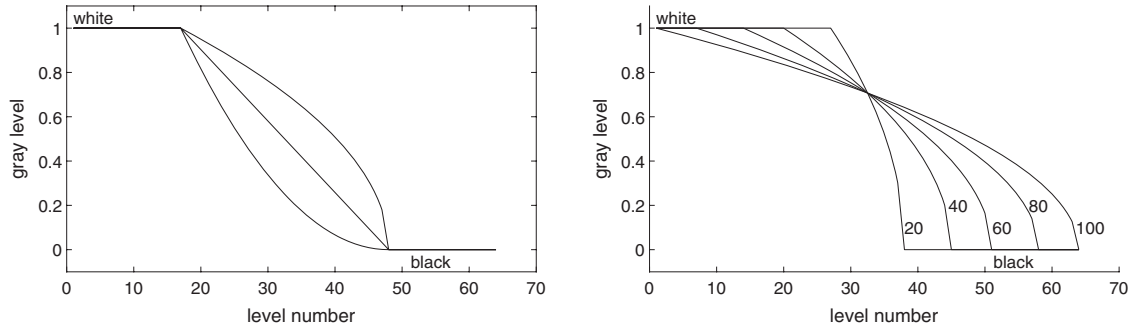


Figure 1.9a (left) A 50% gray curve from `seisclrs` (center) and its brightened (top) and darkened (bottom) versions.
Figure 1.9b (right) Various gray-level curves from `seisclrs` for different `gray_pct` values.

scheme would ramp linearly from 1 (white) at level 1 to 0 (black) at level 64. However, this was found to produce overly dark images with little event standout. Instead, `seisclrs` assigns a certain percentage, called `gray_pct`, of the 64 levels to the transition from white to black and splits the remaining levels between solid black and solid white. For example, by default this percentage is 50, which means that levels 1 through 16 are solid white, 17 through 48 transition from white to black, and 49 through 64 are solid black. The central curve in Figure 1.9a illustrates this. As a final step, `seisclrs` brightens the curve using `brighten` to produce the upper curve in Figure 1.9a. Figure 1.9b shows the gray scales that result from varying the value of `gray_pct`.

Given a gray-level or color scheme, the function `plotimage` has two different data-scaling schemes for determining how seismic amplitudes are mapped to the color bins. The simplest method, called *maximum scaling*, determines the maximum absolute value in the seismic matrix and assigns this the color black, and the negative of this number is white. All other values map linearly into bins in between. This works well for synthetics or for well-balanced real data, but often disappoints for noisy or raw data because of the data's very large dynamic range. The alternative, called *mean scaling*, measures both the mean, $\bar{s}$, and standard deviation, σ_s, of the data and assigns the mean to the center of the gray scale. The ends of the scale are assigned to the values $\bar{s} \pm c\sigma_s$, where c is a user-chosen constant called the *clip level*. Data values outside this range are assigned to the first or last gray-level bin, and are thereby clipped. Thus, mean scaling centers the gray scale on the data mean and the extremes of the scale are assigned to a fixed number of standard deviations from the mean. Larger values of the clip level c correspond to less clipping. In both of these schemes, neutral gray corresponds to zero (if the `seisclrs` color map is used).

Figure 1.10a displays a raw shot record using the maximum-scaling method with `plotimage`. (The shot record is contained in the file *smallshot.mat*, and the traces used in Figure 1.3 are the first and last trace of this record.) The strong energy near the shot sets the scaling levels and is so much stronger than anything else that most other samples fall into neutral gray in the middle of the scale. On the other hand, Figure 1.10b uses mean scaling (and very strong clipping) to show much more of the character of the data.

Code Snippet 1.4.5 This example illustrates the behavior of the *seisclrs* color map. Figures 1.9a and 1.9b are created.

```
figure;
global NOBRIGHTEN

NOBRIGHTEN=1;
s=seisclrs(64,50); %make a 50% linear gray ramp
sb=brighten(s,.5); %brighten it
sd=brighten(s,-.5); %darken it
plot(1:64,[s(:,1) sb(:,1) sd(:,1)],'k')
axis([0 70 -.1 1.1])
text(1,1.05,'white');text(50,-.02,'black')
xlabel('level number');ylabel('gray level');
prepfig
bigfont(gca,1.7,1)
print -depsc .\intrographics\intro5

figure; NOBRIGHTEN=0;
for k=1:5
   pct=max([100-(k-1)*20,1]);
   s=seisclrs(64,pct);
   line(1:64,s(:,1),'color','k');
   if(rem(k,2))tt=.1;else;tt=.2;end
   xt=near(s(:,1),tt);
   text(xt(1),tt,int2str(pct))
end
axis([0 70 -.1 1.1])
text(1,1.05,'white');text(50,-.02,'black')
xlabel('level number');ylabel('gray level');
prepfig
bigfont(gca,1.7,1)
print -depsc .\intrographics\intro5a
```

End Code

introcode / intro5 .m

In addition to the user interface controls in its window, the behavior of *plotimage* can be controlled through *global variables*. Most variables defined in MATLAB are *local* and have a scope that is limited to the setting in which they are defined. For example, a variable called x defined in the base workspace (i.e., at the MATLAB prompt ≫) can only be referenced by a command issued in the base workspace. Thus, a function such as *plotimage* can have its own variable x that can be changed arbitrarily without affecting the x in the base workspace. In addition, the variables defined in a function are transient, meaning that they are erased after the function ends. Global variables are an exception to these rules. Once declared (either in the base workspace or in a function), they remain in existence until explicitly cleared. If, in the base workspace, x is declared to be global, then the x in *plotimage* is still independent unless *plotimage* also declares x global. Then, both the base workspace and *plotimage* address the same memory locations for x and changes in one affect the other.

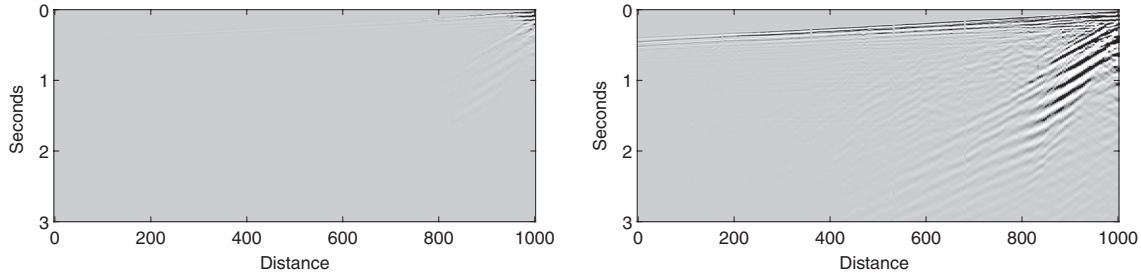


Figure 1.10a (left) A raw shot record displayed using `plotimage` and *maximum scaling*.

Figure 1.10b (right) The same raw shot record as Figure 1.10a but displayed using *mean scaling* with a clip level of 1.

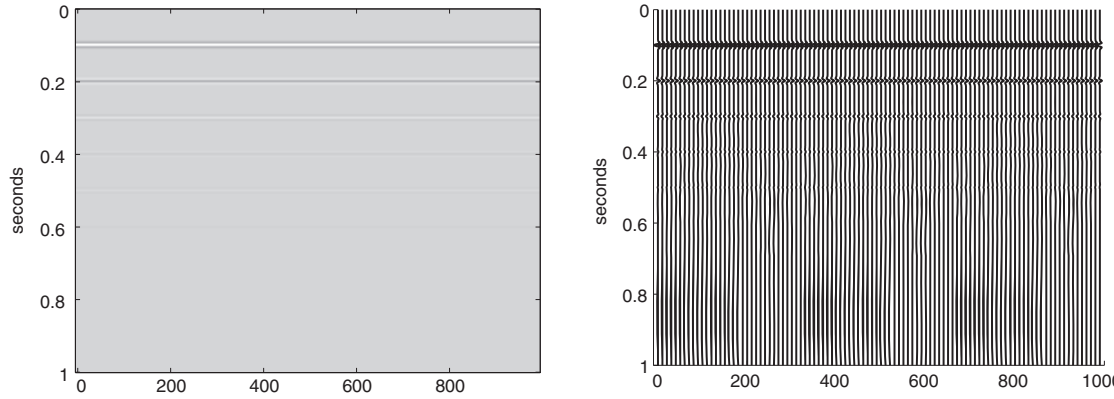


Figure 1.11a (left) A synthetic created to have a 6 dB decrease, with each event displayed by `plotimage`. The event at 0.1 s is full strength and subsequent events occur every 0.1 s. Each event is 6 dB weaker than the preceding event. See Code Snippet 1.4.7.

Figure 1.11b (right) The same synthetic as in Figure 1.11a displayed with `plotseis`. Both plots are displayed without clipping. A visual assessment leads to the conclusion that the dynamic range of `plotimage` is about 24 dB, while that of `plotseis` is perhaps 18 dB.

Code Snippet 1.4.6 illustrates the assignment of global variables for `plotimage` as done in the author's startup.m file. These variables only determine the state of a `plotimage` window at the time it is launched. The user can always manipulate the image as desired using the user interface controls. Any such manipulations will not affect the global variables, so that the next `plotimage` window will launch in exactly the same way.

The dynamic ranges of image and WTVA displays are generally different. When both are displayed without clipping using a moderate trace spacing (Figures 1.11a and 1.11b), the dynamic range of `plotimage` ($\approx$24 dB) tends to be slightly greater than that of `plotseis` ($\approx$18 dB). However, the behavior of `plotimage` is nearly independent of the trace density, while `plotseis` becomes useless at high densities.

Code Snippet 1.4.6 This is a portion of the author's *startup.m* file that illustrates the setting of global variables to control *plotimage*.

```
% save this to your Matlab directory and rename to startup.m
global SCALE_OPT GRAY_PCT NUMBER_OF_COLORS
global CLIP COLOR_MAP NOBRIGHTEN NOSIG
global BIGFIG_X BIGFIG_Y BIGFIG_WIDTH BIGFIG_HEIGHT
%set parameters for plotimage
SCALE_OPT=2;
GRAY_PCT=20;
NUMBER_OF_COLORS=64;
CLIP=4;
COLOR_MAP='seisclrs';
NOBRIGHTEN=1;
NOSIG=1;
% set parameters for bigfig (used by prepfig)
% try to make the enlarged figure size 1100x700 pixels
scr=get(0,'screensize');
BIGFIG_X=1;
BIGFIG_Y=30;
if(scr(3)>1100)
    BIGFIG_WIDTH=1100;
else
    BIGFIG_WIDTH=scr(3)-BIGFIG_X;
end
if(scr(4)>730)
    BIGFIG_HEIGHT=700;
else
    BIGFIG_HEIGHT=scr(4)-BIGFIG_Y;
end
%
% By default, your working directory will be documents\matlab
% (under Windows) Startup.m should reside in this directory.
% If you wish to always begin working in another directory,
% for example \documents\matlab\work, then uncomment the
% following line
% cd work
```

End Code

introcode / sample_startup .m

Exercises

1.4.4 Load the file `smallshot.mat` into your base workspace using the command `load` and display the shot record with *plotimage*. Study the online help for *plotimage* and define the six global variables that control its behavior. Practice changing their values and observe the results. In particular, examine some of the color maps: hsv, hot, cool, gray, bone, copper, pink, white, flag, jet, winter,

Code Snippet 1.4.7 This code creates a synthetic with a 6 dB decrease for each event and displays it without clipping with both *plotimage* and *plotseis*. See Figures 1.11a and 1.11b.

```
% Put an event every 100 ms.
% Each event decreases in amplitude by 6 db.
r=zeros(501,1);dt=.002;ntr=100;
t=(0:length(r)-1)*dt;
amp=1;
for tk=.1:.1:1
   k=round(tk/dt)+1;
   r(k)=(-1)^(round(tk/.1))*amp;
   amp=amp/2;
end
w=ricker(.002,60,.1);
s=convz(r,w);
seis=s*ones(1,ntr);
x=10*(0:ntr-1);
global SCALE_OPT GRAY_PCT
SCALE_OPT=2;GRAY_PCT=100;
plotimage(seis,t,x);ylabel('seconds')
hax=findobj(gca,'tag','MAINAXES');
pos=get(hax,'position');
nudge=.05;
set(hax,'position',[pos(1) pos(2)+nudge pos(3) pos(4)-nudge]);
title('')
prepfig
bigfont(gcf,1,1);whitefig;xlabel('')
hideui;

print -depsc .\intrographics\intro7

figure
plotseis(seis,t,x,1,1,1,1,'k');ylabel('seconds')
set(gca,'xaxislocation','bottom')
%plotseismic(seis,t,x);
title('')
prepfig
bigfont(gcf,1,1)

print -depsc .\intrographics\intro7a
```

End Code

introcode / intro6 .m

spring, summer, and autumn. (Be sure to specify the name of the color map inside single quotes.)

1.4.5 Recreate Figures 1.11a and 1.11b using Code Snippet 1.4.7 as a guide. Experiment with different numbers of traces (`ntr`) and different program settings to see how the dynamic range is affected.

1.4.3 The `plotimage` Picking Facility

The function *`plotimage`*, in addition to displaying seismic data, provides a rudimentary seismic *picking* facility. This is similar in concept to using MATLAB's `ginput` (described in the next section) but is somewhat simpler to use. In concept, this facility simply allows the user to draw any number of straight-line segments (picks) on top of the data display and affords easy access to the coordinates of these picks. The picks can then be used in any subsequent calculation or analysis.

The picking mechanism is activated by the popup menu in the lower left corner of the *`plotimage`* window. Initially, in the *zoom* setting, this menu has two picking settings: *Pick(N)* and *Pick(O)*. The *N* and *O* stand for *New* and *Old*, indicating whether the picks about to be made are a new list or should be added to the existing list. The pick list is stored as the global variable `PICKS`, which can be accessed from the base workspace or within any function by declaring it there. There is only one pick list, regardless of how many *`plotimage`* windows are open. This means that if one *`plotimage`* window has been used for picking and a second is then activated with the *Pick(N)* option, the picks for the first window will be lost. If desired, this can be avoided by copying the pick list into another variable prior to activating the second picking process.

A pick is made by clicking the left mouse button on the first point of the pick, holding the button down while dragging to the second point, and releasing the button. The pick is drawn on the screen in a temporary color and then drawn in a permanent color when the mouse button is released. The permanent color is controlled by the global variable `PICKCOLOR`. If this variable has not been set, picks are drawn in red. A single mouse click, made without dragging the mouse and within the bounds of the axes, will delete the most recent pick.

This picking facility is rudimentary in the sense that the picks are established without any numerical calculation involving the data. Furthermore, there is no facility to name a set of picks or to retain their values. Nevertheless, a number of very useful calculations, most notably raytrace migration (see Sections 7.2.2 and 7.3.3), are enabled by this. It is also quite feasible to implement a more complete picking facility on this base.

1.4.4 Picked Events and Drawing on Top of Seismic Data

Often it is useful to draw lines or even filled polygons on top of a seismic data display. This is quite easily done using MATLAB's `line` and `patch` commands. Only `line` is discussed here. To indicate the motivation behind the need to draw upon a seismic display, we begin with a general discussion of picking.

The concept of *picking* seismic data refers to a general procedure of selecting the arrival times and spatial positions of seismic events of interest and then, optionally, extracting some attribute from the data at the picked locations. The most obvious attribute is simply the event amplitude at the pick location. Other popular attributes include the *Hilbert attributes* (which are the envelope, instantaneous phase, and instantaneous frequency; see Section 2.4.6), the apparent velocity of the event, and many others. Here we will consider only the event time and demonstrate the picking of the first arrivals, or first breaks, at each

receiver on a shot record. The actual mechanism of picking ranges from manual to fully automatic picking. Here we will use automatic picking as performed by `picker`.

Code Snippet 1.4.8 illustrates the use of `picker` to pick the first breaks of a shot record. The first three input arguments to `picker` are simply the shot matrix, `seis`, and its time and space coordinate vectors, `t` and `x`. The most important inputs are the next four. The event of interest is defined by `te` and `xe`, which are identically sized vectors giving the times and spatial positions of points defining the approximate trajectory of the event. These need not be precise but should define a segmented line that roughly follows the event within, say, 0.01 s. Since, in this case, the first break is essentially linear, it is sufficient to specify a single point at the beginning and end of the event. The sixth input, `delt`, specifies the half-width of a fairway centered on the event and within which the picks are constrained to lie. Thus, in this case the event is defined to extend from $(t, x) = (0.4, 0.0)$ to $(t, x) = (0.02, 1000)$ and the fairway has width 0.4 (seconds). The final input, `picktype`, is a flag that specifies the type of picking to be performed. At the time of writing, the following `picktypes` are supported:

- 1: pick the maximum absolute value of the amplitude in the fairway.
- 1.1: pick the maximum of the Hilbert envelope in the fairway.
- 2: pick the closest (to the event) peak of the Hilbert envelope in the fairway.
- 3: pick the closest (to the event) peak in the fairway.
- 4: pick the closest (to the event) trough in the fairway.
- 5: pick the closest (to the event) + to − zero crossing in the fairway.
- 6: pick the closest (to the event) − to + zero crossing in the fairway.
- 7: pick the closest (to the event) zero crossing of either polarity.
- 8: pick first breaks by sta(env)/lta(env) > threshold.
- 9: pick first breaks by the maximum of the Hilbert envelope in the fairway.

In Code Snippet 1.4.8, `picktype` takes the value 8, which indicates that we are doing first-break picking, and the cryptic description "sta(env)/lta(env) > threshold" means that the onset of the first break is indicated when the ratio of a *short-term average*, or sta, to a *long-term average*, or lta, exceeds a defined threshold. These averages are calculated not from the trace itself but from its enclosing Hilbert envelope. To produce the result in Figure 1.12a, the default values for "short-term" and "long-term" were used. By default, the lta for each sample is computed over a window which begins at the sample and extends to later times for a length that is 10% of the fairway. The sta is computed similarly but the window size is 10% of that used for the lta. Before computing these averages, the portion of the trace enclosed in the fairway is resampled to 10 times more samples so that the picked times are easily estimated to one-tenth of the time-sample interval of the original trace. The actual pick computation uses the ratio

$$r = \frac{g(t)e_{\text{sta}}(t)}{e_{\text{lta}}(t) + \lambda e_{\max}}, \tag{1.2}$$

where e_{sta} and e_{lta} refer to the short-term and long-term averages of the Hilbert envelope, $e_{\max}$ is the maximum of e_{lta}, $g(t)$ is a Gaussian window centered on the event time with

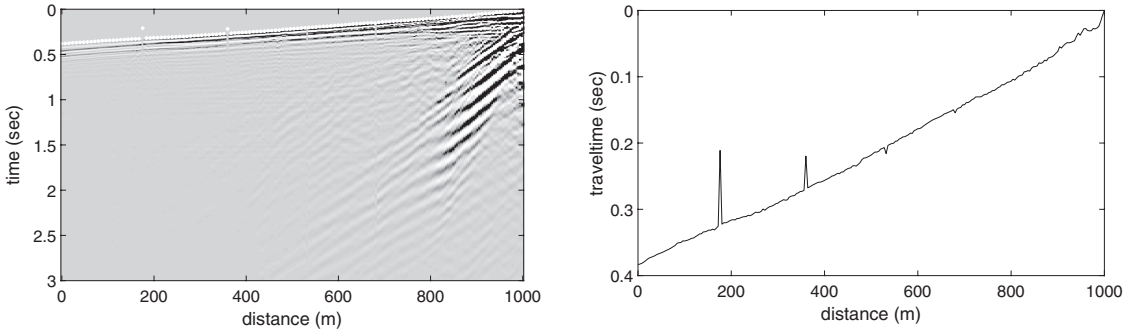


Figure 1.12a (left) This figure was created by Code Snippet 1.4.8 and used the automatic picking function *picker* to pick the first breaks. The white dots denote the picked first-break times.

Figure 1.12b (right) The picked times from the first breaks of Figure 1.12a. The anomalous outliers occur on traces that are themselves anomalous. These usually correspond to malfunctioning or poorly planted geophones.

Code Snippet 1.4.8 This example loads *smallshot.mat* and uses `picker` to pick the first breaks (lines 3 and 4). After the data are plotted with *plotimage*, the picked locations of the first breaks are drawn on top of the seismic record using *line*. Finally, the pick times are plotted versus receiver position. The results are shown in Figures 1.12a and 1.12b.

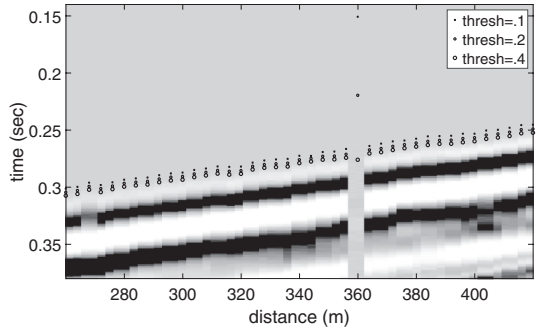
```
1   load smallshot
2   global THRESH
3
4   te=[.4 .02];xe=[0 1000];delt=.2;picktype=8;
5   [ap,ae,tpick,xpick]=picker(seis,t,x,te,xe,delt,picktype);
6   THRESH=.1;
7   [ap2,ae2,tpick2,xpick2]=picker(seis,t,x,te,xe,delt,picktype);
8   THRESH=.4;
9   [ap3,ae3,tpick3,xpick3]=picker(seis,t,x,te,xe,delt,picktype);
10
11  plotimage(seis,t,x);title('')
12  xlabel('distance (m)');ylabel('time (sec)')
13  h=line(xpick(1:2:end),tpick(1:2:end),'markeredgecolor','w',...
14      'linestyle','none','marker','.','markersize',8);
15
16  figure;plot(xpick,tpick,'k');flipy
17  xlabel('distance (m)');ylabel('traveltime (s)')
18
```

End Code

introcode / pick_fb .m

standard deviation of one quarter of the fairway width, and $0 < \lambda < 1$ is a small positive number. Given the computation of r, a pick is made at the earliest time where

$$r > r_{\text{thresh}}, \tag{1.3}$$



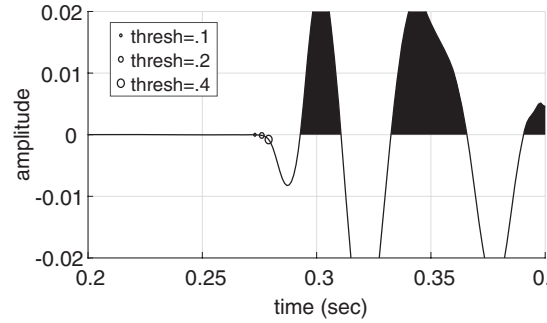


Figure 1.13a (left) An enlargement of a portion of the data of Figure 1.12a showing the first-break picks for three different threshold values.

Figure 1.13b (right) The trace at $x = 340$ m from Figure 1.13a is shown together with its first-break picks for three different thresholds.

where r_{thresh} defaults to 0.2. In Code Snippet 1.4.8, the first breaks are picked three times with different values of the threshold. The first time uses the default value of 0.2, the second time uses a smaller value of 0.1, and the third time uses a larger value of 0.4. Figure 1.12a shows the default threshold results, while Figures 1.13a and 1.13b compare all three thresholds.

Equation (1.2) contains two significant modifications to the simple sta–lta ratio. First, the denominator is expressed as $e_{\text{lta}}(t) + \lambda e_{\max}$ instead of just $e_{\text{lta}}(t)$, and, second, $g(t)$ is present in the numerator. Denominator modification is a very common technique that is used in many settings where a numerical division involving real data must be performed. With real data, a signal strength may fall to low levels for unknown reasons that lie wholly outside whatever theoretical concepts are being employed. When used as a divisor, these low signal levels give very large results that are usually false indicators. The modification used here prevents the denominator from getting "too small" and the choice of the number λ determines the meaning of "too small." The second modification is to multiply the ratio by a Gaussian function, or window, given by

$$g(t) = e^{(-(t-te)^2/\sigma^2)}, \tag{1.4}$$

where te is the event time and σ is the standard deviation, which is chosen to be one quarter of the width of the fairway. As expressed, $g(t)$ will be unity at the event time and decreases by two standard deviations to about 0.0183 at the edge of the fairway. This is an example of introducing a deliberate bias that forces the picks to be near the defined event time.

The call to `line` in Code Snippet 1.4.8 plots the points on top of the seismic image. The first two input arguments are the coordinates of the line to be drawn. The return value from `line` is called the *graphics handle* of the line and allows the properties of the lines to be interrogated or set. In addition to the line coordinates, `line` accepts an arbitrary-length list of (attribute, property) pairs that define various features of the line. In this case, `markeredgecolor` is set to *white*, its line style is set to *none*, markers

are assigned to each data point as simple dots, and the marker size is set to 8. The net result is that we see isolated white dots for each first-break pick with no connecting line. There are many other possible attributes that can be set in this way. To see a list of them, issue the command `get(h)`, where `h` is the handle of a line, and MATLAB will display a complete property list. You can also issue `set(h)` to see a list of parameters with their legal values. Once the line has been drawn, you can alter any of its properties with the `set` command. For example, `set(h,'markeredgecolor','c')` changes the marker color to cyan and `set(h,'ydata',get(h,'ydata')+.1)` shifts the line down by 0.1 s.

Figures 1.13a and 1.13b examine the effect of the threshold value on the picking. Lowering the threshold shifts the pick to an earlier time but runs the risk of introducing more false picks on noisy or weak traces such as the trace at $x = 360$ m. Increasing the threshold moves the picks to a later time while reducing the chances of noisy data causing erroneous picks. In Figure 1.13a, there is only one problematic pick, at position $x = 360$ m, while in Figure 1.12a there are two more obvious problems at $x = 176$ m and $x = 532$ m. These all appear to be associated with weak traces and are probably noise dominated. The pick at $x = 360$ m is clearly more consistent with its neighbors when a larger threshold is used.

1.5 Programming Tools

It is recommended that a study course based on this book should involve a considerable amount of programming. Programming in MATLAB has similarities to other languages but also has its unique quirks. This section discusses some strategies for using MATLAB effectively to explore and manipulate seismic datasets. MATLAB has two basic programming constructs, *scripts* and *functions*, and *program* will be used to refer to both.

1.5.1 Scripts

MATLAB is designed to provide a highly interactive environment that allows very rapid testing of ideas. However, unless some care is taken, it is very easy to create results that are nearly irreproducible. This is especially true if a long sequence of complex commands has been entered directly at the MATLAB prompt. A much better approach is to type the commands into a text file and execute them as a *script*. This has the virtue that a permanent record is maintained, so that the results can be reproduced at any time by simply reexecuting the script.

A script is the simplest possible MATLAB programming structure. It is nothing more than a sequence of syntactically correct commands that have been typed into a file. The file name must end in .m and must appear in the MATLAB search path. When you type the file name (without the .m) at the MATLAB prompt, MATLAB searches its path for the first .m file with that name and executes the commands contained therein. If you are in doubt

about whether your script is the first so-named file in the path, use the `which` command. Typing `which foo` causes MATLAB to display the complete path to the file that it will execute when `foo` is typed.

A good practice is to maintain a folder in MATLAB's search path that contains the scripts associated with each logically distinct project. These script directories can be managed by creating a further set of control scripts that appropriately manipulate MATLAB's search path. For example, suppose that most of your MATLAB tools are contained in the directory

```
C:\Matlab\toolbox\local\mytools
```

and that you have project scripts stored in

```
C:\My Documents\project1\scripts
```

and

```
C:\My Documents\project2\scripts.
```

Then, a simple management scheme is to include something like

```
1  global MYPATH
2  if(isempty(MYPATH))
3     p=path;
4     path([p ';C:\MatlabR11\toolbox\local\mytools']);
5     MYPATH=path;
6  else
7     path(MYPATH);
8  end
```

in your startup.m file and then create a script called project1.m that contains

```
1  p=path;
2  path([p ';C:\My Documents\project1\scripts'])
```

and a similar script for project2. When you launch MATLAB, your startup.m establishes your base search path and assigns it to a global variable called `MYPATH`. Whenever you want to work on project1, you simply type `project1` and your base path is altered to include the project1 scripts. Typing `startup` again restores the base path, and typing `project2` sets the path for that project. Of course, you could just add all of your projects to your base path, but then you would not be able to have scripts with the same name in each project.

Exercises

1.5.1 Write a script that plots wiggle traces on top of an image plot. Test your script with the synthetic section of Code Snippet 1.4.4. (Hint: *`plotseis`* allows its plot to be directed to an existing set of axes instead of creating a new one. The "current" axes can be referred to with `gca`.)

1.5.2 Functions

The MATLAB *function* provides a more advanced programming construct than the script. The latter is simply a list of MATLAB commands that execute in the base workspace. Functions execute in their own independent workspace, which communicates with the base workspace through *input and output variables*.

The Structure of a Function

A simple function that clips a seismic trace is given in Code Snippet 1.5.1. This shows more detail than will be shown in other examples in this book in order to illustrate function documentation. The first line of *clip* gives the basic syntax for a function declaration. Since both scripts and functions are in .m files, it is this line that alerts MATLAB to the fact that *clip* is not a script. The function is declared with the basic syntax

`[output_variable_list]=function_name(input_variable_list).`

Rectangular brackets [. . .] enclose the list of output variables, while regular parentheses enclose the input variables. Either list can be of any length, and entries are separated by commas. For *clip*, there is only one output variable and the [. . .] are not required. When the function is invoked, either at the command line or in a program, a choice may be made to supply only the first *n* input variables or to accept only the first *m* output variables. For example, given a function definition like `[x1,x2]=foo(y1,y2)`, it may be legally invoked with any of

```
[a,b]=foo(c,d);
[a,b]=foo(c);
a=foo(c,d);
a=foo(c);
```

The variable names in the function declaration have no relation to those actually used when the function is invoked. For example, when `a=foo(c)` is issued, the variable `c` becomes the variable `(y1)` within `foo`'s workspace and, similarly, `foo`'s `x1` is returned to become the variable `a` in the calling workspace. Though it is possible to call `foo` without specifying `y2`, there is no simple way to call it and specify `y2` while omitting `y1`. Therefore, it is advisable to structure the list of input variables such that the most important ones appear first. If an input variable is not supplied, then it must be assigned a *default* value by the function.

In the example of the function *clip*, there are two input variables, both mandatory, and one output variable. The next few lines after the function definition are all comments (i.e., nonexecutable), as is indicated by the % sign that precedes each line. The first contiguous block of comments following the function definition constitutes the *online help*. Typing `help clip` at the MATLAB prompt will cause these comment lines to be echoed to your MATLAB window.

The documentation for *clip* follows a simple but effective style. The first line gives the function name and a simple synopsis. Typing `help directory_name`, where `directory_name` is the name of a directory containing many .m files, causes the first line

Code Snippet 1.5.1 A simple MATLAB function to clip a signal.

```
1   function trout=clip(trin,amp);
2   % CLIP performs amplitude clipping on a seismic trace
3   %
4   % trout=clip(trin,amp)
5   %
6   % CLIP adjusts only those samples on trin which are greater
7   % in absolute value than 'amp'. These are set equal to amp but
8   % with the sign of the original sample.
9   %
10  % trin= input trace
11  % amp= clipping amplitude
12  % trout= output trace
13  %
14  % by G.F. Margrave, May 1991
15  %
16  if(nargin~=2)
17     error('incorrect number of input variables');
18  end
19  % find the samples to be clipped
20   indices=find(abs(trin)>amp);
21  % clip them
22   trout=trin;
23   trout(indices)=sign(trin(indices))*amp;
```

End Code

introcode / clip_example .m

of each file to be echoed, giving a summary of the directory. Next appears one or more *function prototypes* that give examples of correct function calls. After a function's help file has been viewed, a prototype can be copied to the command line and edited for the task at hand. The next block of comments gives a longer description of the tasks performed by the function. Then comes a description of each input and output parameter and their defaults, if any. Finally, it is good form to put your name and the date at the bottom. If your code is to be used by anyone other than you, this is valuable because it establishes who owns the code and who is responsible for fixes and upgrades.

Following the online help is the body of the code. Lines 16–18 illustrate the use of the automatically defined variable `nargin`, which is equal to the number of input variables supplied at calling. *`clip`* requires both input variables, so it aborts if `nargin` is not 2 by calling the function `error`. The assignment of *default values* for input variables can also be done using `nargin`. For example, suppose the `amp` parameter were to be allowed to default to 1.0. This can be accomplished quite simply with

```
if(nargin<2) amp=1; end
```

Lines 20, 22, and 23 actually accomplish the clipping. They illustrate the use of vector addressing and will be discussed more thoroughly in Section 1.6.1.

Code Snippet 1.5.2 Assume that a matrix `seis` already exists and that `nzs` and `nzt` are already defined as the sizes of zero pads in samples and traces, respectively. Then a padded seismic matrix is computed as follows:

```
1   %pad with zero samples
2   [nsamp,ntr]=size(seis);
3   seis=[seis;zeros(nzs,ntr)];
4   %pad with zero traces
5   seis=[seis zeros(nsamp+nzs,nzt)];
```

End Code

introcode / introerror .m

1.5.3 Coping with Errors and the MATLAB Debugger

No matter how carefully you work, eventually you will produce code with errors in it. The simplest errors are usually due to mangled syntax and are relatively easy to fix. A MATLAB language guide is essential for the beginner to decipher the syntax errors.

More subtle errors only become apparent at run time, after correcting all of the syntax errors. Very often, run-time errors are related to incorrect matrix sizes. For example, suppose we wish to pad a seismic matrix with `nzs` zero samples on the end of each trace and `nzt` zero traces on the end of the matrix. The correct way to do this is shown in Code Snippet 1.5.2.

The use of [. . .] on lines 3 and 5 is key here. Unless they are being used to group the return variables from a function, square brackets [. . .] generally indicate that a new matrix is being formed from a concatenation of existing matrices. On line 3, `seis` is concatenated with a matrix of zeros that has the same number of traces as `seis` but `nzs` samples. The semicolon between the two matrices is crucial here, as it is the *row separator*, while a space or comma is the *column separator*. If the elements in the brackets on line 3 were separated by a space instead of a semicolon, MATLAB would emit the error message

```
Error using horzcat
Dimensions of matrices being concatenated are not consistent.
```

This occurs because the use of the column separator tells MATLAB to put `seis` and `zeros(nxs,ntr)` side by side in a new matrix; but this is only possible if the two items have the same number of rows. Similarly, if a row separator is used on line 5, MATLAB will complain that `Error using vertcat Dimensions of matrices being concatenated are not consistent`.

Another common error is an assignment statement in which the matrices on either side of the equals sign do not evaluate to matrices of the same size. This is a fundamental requirement and calls attention to the basic MATLAB syntax `MatrixA = MatrixB`. That is, the entities on both sides of the equals sign must evaluate to matrices of exactly the same size. The only exception to this is that the assignment of a constant to a matrix is allowed.

One rather insidious error deserves special mention. MATLAB allows variables to be "declared" by simply using them in context in a valid expression. Though convenient, this can cause problems. For example, if a variable called `plot` is defined, then it will mask the command `plot`. All further commands to plot data (e.g., `plot(x,y)`) will be interpreted as indexing operations into the matrix `plot`. More subtly, if the letter `i` is used as the index of a loop, then it can no longer serve its predefined task as $\sqrt{-1}$ in complex arithmetic. Therefore some caution is called for in choosing variable names, and, especially, the common Fortran practice of choosing `i` as a loop index is to be discouraged.

The most subtle errors are those which never generate an overt error message but still cause incorrect results. These logical errors can be very difficult to eliminate. Once a program executes successfully, it must still be verified that it has produced the expected results. Usually this means running it on several test cases whose expected behavior is well known, or comparing it with other codes whose functionality overlaps with the new program.

The MATLAB debugger is a very helpful facility for resolving run-time errors, and it is simple to use. There are just a few essential debugger commands, such as `dbstop`, `dbstep`, `dbcont`, `dbquit`, `dbup`, and `dbdown`. A debugging session is typically initiated by issuing the `dbstop` command to tell MATLAB to pause execution at a certain line number, called a *breakpoint*. For example, `dbstop at 200 in plotimage` will cause execution to pause at the executable line nearest line 200 in *`plotimage`*. At this point, you may choose to issue another debugger command, such as `dbstep`, which steps execution a line at a time, or you may issue any other valid MATLAB command. This means that the entire power of MATLAB is available to help discover the problem. Especially when dealing with large datasets, it is often very helpful to issue plotting commands to graph the intermediate variables.

As an example of the debugging facility, consider the problem of extracting a *slice* from a matrix. That is, given a 2D matrix and a trajectory through it, extract a submatrix consisting of those samples within a certain half-width of the trajectory. The trajectory is defined as a vector of row numbers, one per column, that cannot double back on itself. A first attempt at creating a function for this task might be like that in Code Snippet 1.5.3.

First, prepare some simple data like this:

```
» a=((5:-1:1)')*(ones(1,10))

a =

     5     5     5     5     5     5     5     5     5     5
     4     4     4     4     4     4     4     4     4     4
     3     3     3     3     3     3     3     3     3     3
     2     2     2     2     2     2     2     2     2     2
     1     1     1     1     1     1     1     1     1     1

» traj=[1:5 5:-1:1]
traj =
     1     2     3     4     5     5     4     3     2     1
```

Code Snippet 1.5.3 Here is some code for slicing through a matrix along a trajectory. Beware: this code generates an error.

```
function s=slicem(a,traj,hwid)

[m,n]=size(a);
for k=1:n %loop over columns
        i1=max(1,traj(k)-hwid); %start of slice
        i2=min(m,traj(k)+hwid); %end of slice
        ind=(i1:i2)-traj(k)+hwid; %output indices
        s(ind,k) = a(i1:i2,k); %extract the slice
end
```

End Code

introcode / slicem . m

The samples of `a` that lie on the trajectory are `5 4 3 2 1 1 2 3 4 5`. Executing `slicem` with the command `s=slicem(a,traj,1)` generates the following error:

```
Subscript indices must either be real positive integers or logicals.

Error in ==> slicem.m On line 8  ==>  s(ind,k) = a(i1:i2,k);
```

This error message suggests that there is a problem with indexing on line 8; however, this could be occurring in indexing into either `a` or `s`. To investigate, issue the command `dbstop at 8 in slicem` and rerun the program. Then MATLAB stops at line 8 and prints

```
8       s(ind,k) = a(i1:i2,k);
K≫
```

Now, suspecting that the index vector `ind` is at fault, we list it to see that it contains the values `1 2` and so does the vector `i1:i2`. These are legal indices. Noting that line 8 is in a loop, it seems possible that the error may occur in some further iteration of the loop (we are at `k` = 1). So, execution is resumed with the command `dbcont`. At `k` = 2, we discover that `ind` contains the values `0 1 2`, while `i1:i2` is `1 2 3`. Thus it is apparent that the vector `ind` is generating an illegal index of `0`. A moment's reflection reveals that line 7 should be coded as `ind=(i1:i2)-traj(k)+hwid+1;`, which includes an extra `+1`. Therefore, we exit from the debugger with `dbquit`, make the change, and rerun *`slicem`* to get the correct result:

```
s =

     0     5     4     3     2     2     3     4     5     0
     5     4     3     2     1     1     2     3     4     5
     4     3     2     1     0     0     1     2     3     4
```

The samples on the trajectory appear in the central row, while the other rows contain neighboring values unless such values exceed the bounds of the matrix `a`, in which case a zero is returned. A more polished version of this code is found as *`slicemat`* in the *NMES Toolbox*. Note that a simpler debugger procedure (rather than stopping at line 8) would

be to issue the command `dbstop if error`, but the process illustrated here shows more debugging features.

1.6 Programming for Efficiency

1.6.1 Vector Addressing

The actual trace clipping in Code Snippet 1.5.1 is accomplished by three deceptively simple lines. These lines employ an important MATLAB technique called *vector addressing* that allows arrays to be processed without explicit loop structures. This is a key technique to master in order to write efficient MATLAB code. Vector addressing means that MATLAB allows an index into an array to be a vector (or, more generally, a matrix) while languages like C and Fortran allow only scalar indices. So, for example, if `a` is a vector of length 10, then the statement `a([3 5 7])=[pi 2*pi sqrt(pi)]` sets the third, fifth, and seventh entries to π, $2 * \pi$, and $\sqrt{\pi}$, respectively. In *`clip`*, the function `find` performs a logical test and returns a *vector of indices* pointing to samples that satisfy the test. The next line creates the output trace as a copy of the input. Finally, the last line uses vector addressing to clip those samples identified by `find`.

In contrast to the vector coding style just described, Code Snippet 1.6.1 shows how *`clip`* might be coded by someone stuck in the rut of scalar addressing (a C or Fortran programmer). This code is logically equivalent to *`clip`* but executes much slower. This is easily demonstrated using MATLAB's built-in timing facility. Code Snippet 1.6.2 is a simple script that uses the `tic` and `toc` commands for this purpose. `tic` sets an internal timer, and `toc` writes out the elapsed time since the previous `tic`. The loops are executed 100 times to allow minor fluctuations to average out. On the second execution of this script, MATLAB responds with

```
elapsed_time = 0.0500
elapsed_time = 1.6000
```

which shows that *`clip`* is 30 times faster than *`fortclip`*. (On the first execution, *`clip`* is only about 15 times faster because MATLAB spends some time doing internal compiling. Run-time tests should always be done a number of times to allow for effects like this.)

A simple blunder that can slow down *`fortclip`* even more is to write it like Code Snippet 1.6.3. This version of *`fortclip`* still produces correct results, but is almost 50 times slower than *`clip`*. The reason is that the output trace, `trout`, is being addressed sample by sample but has not been preallocated. This forces MATLAB to resize the vector each time through the loop. Such resizing is slow and may require MATLAB to make repeated requests to the operating system to grow its memory.

Code Snippet 1.6.1 A Fortran programmer new to MATLAB would probably code *clip* in this way.

```
function trout=fortclip(trin,amp)

for k=1:length(trin)
   if(abs(trin(k)>amp))
      trin(k)=sign(trin(k))*amp;
   end
end

trout=trin;
```

End Code

introcode / fortclip .m

Code Snippet 1.6.2 This script compares the execution times of *clip* and *fortclip*.

```
[r,t]=reflec(1,.002,.2);

tic
for k=1:100
   r2=clip(t,.05);
end
toc

tic
for k=1:100
   r2=fortclip(t,.05);
end
toc
```

End Code

introcode / introclip .m

Exercises

1.6.1 Create a version of *fortclip* and verify the run-time comparisons quoted here. Your computer may give different results. Show that the version of *fortclip* in Code Snippet 1.6.3 can be made to run as fast as that in Code Snippet 1.6.1 by using the `zeros` function to preallocate `trout`.

1.6.2 Vector Programming

It is the authors' experience that MATLAB code written using vector addressing and linear algebra constructs can be quite efficient. In fact, run times can approach those of compiled C or Fortran. On the other hand, coding in the scalar style can produce very slow programs

Code Snippet 1.6.3 An even slower version of *fortclip*.

```
1   function trout=fortclip(trin,amp)
2
3   for k=1:length(trin)
4      if(abs(trin(k)>amp))
5         trout(k)=sign(trin(k))*amp;
6      end
7   end
```

End Code

introcode / fortclipdumb .m

that give correct results but lead to the false impression that MATLAB is a slow environment. MATLAB is designed to eliminate many of the loop structures found in other languages. Vector addressing helps in this regard, but even more important is the use of MATLAB's linear algebra constructs. Programming efficiently in this way is called *vector programming*.

As an example, suppose `seis` is a seismic data matrix and it is desired to scale each trace (i.e., column) by a constant scale factor. Let `scales` be a vector of such factors whose length is equal to the number of columns in `seis`. Using scalar programming, a seasoned Fortran programmer might produce something like Code Snippet 1.6.4.

This works correctly but is needlessly slow. An experienced MATLAB programmer would know that the operation of "matrix times diagonal matrix" results in a new matrix whose columns are scaled by the corresponding elements of the diagonal matrix. You can check this out for yourself with a simple MATLAB exercise:

```
a=ones(4,3); b=diag([1 2 3]); a*b
```

to which MATLAB's response is

```
ans =

     1     2     3
     1     2     3
     1     2     3
     1     2     3
```

Thus the MATLAB programmer would write Code Snippet 1.6.4 with the very simple single line in Code Snippet 1.6.5. Tests indicate that Code Snippet 1.6.5 is about 10 times as fast as Code Snippet 1.6.4.

Vector programming is also facilitated by the fact that MATLAB's mathematical functions are automatically set up to operate on arrays. This includes functions such as `sin`, `cos`, `tan`, `atan`, `atan2`, `exp`, `log`, `log10`, and `sqrt`. For example, if `t` is a time coordinate vector, then `sin(2*pi*10*t)` is a vector of the same size as `t` whose entries are a 10 Hz sine wave evaluated at the times in `t`. These functions all perform element-by-element operations on matrices of any size. Code written in this way is actually more visually similar to the corresponding mathematical formulas than is scalar code with its many explicit loops.

Code Snippet 1.6.4 A Fortran programmer might write this code to scale each trace in the matrix `seis` by a factor from the vector `scales`.

```
1   [nsamp,ntr]=size(seis);
2
3   for col=1:ntr
4      for row=1:nsamp
5         seis(row,col)=scales(col)*seis(row,col);
6      end
7   end
```

End Code

introcode / intro8 .m

Code Snippet 1.6.5 A MATLAB programmer would write the example of Code Snippet 1.6.4 in this way:

```
1   seis=seis*diag(scales);
```

End Code

introcode / intro8a .m

Some of MATLAB's vectorized functions operate on matrices but return matrices one dimension smaller. The functions `sum`, `cumsum`, `mean`, `main`, `max`, and `std` all behave in this manner. For example, if `trace` is a vector and `seis` is a matrix, then `mean(trace)` results in a scalar that is the mean value of the vector, while `mean(seis)` results in a *row vector* containing the mean value of each *column* of `seis`. The mean value of the entire matrix, `seis`, can be calculated with `mean(mean(seis))`.

1.6.3 The Colon Symbol in MATLAB

Vector programming is also facilitated by a thorough understanding of the multiple uses of the colon (:) in MATLAB. In its most elementary form, the colon is used to generate vectors. For example, the time coordinate vector for a trace with `ntr` samples and a sample interval of `dt` can be generated with the command `t=(0:ntr-1)*dt`. This results in a *row vector* whose entries are `[0 dt 2*dt 3*dt ... (ntr-1)*dt]`.

More subtle is the use of the colon in indexing into a matrix. Let `seis` be a two-dimensional seismic matrix. Then some sample indexing operations are the following:

`seis(:,10)` refers to the tenth trace. The colon indicates that all rows (samples) are desired.

`seis(:,50:60)` refers to traces 50 through 60.

`seis(100:500,50:60)` selects samples 100 through 500 on traces 50 through 60.

`seis(520:2:720,:)` grabs every other sample between sample number 520 and number 720 on all traces.

`seis(:,90:-1:50)` selects traces 50 through 90 but in reverse order.

`seis(:)` refers to the entire seismic matrix as a giant column vector.

Of course, the indices need not be entered as explicit numbers but can be a variable instead, as in the case of the return from `find` in Code Snippet 1.5.1.

The last item needs more explanation. Regardless of the dimensions of a matrix, it can always be referred to as a single column vector using a single index. For example, the two-dimensional matrix `seis` can be indexed as `seis(irow,icol)` and as `seis(ipos)`. This is possible because computer memory is actually a one-dimensional space and MATLAB stores a matrix in this space in *column order*. This means that the actual storage pattern of `seis` is `[seis(1,1) seis(2,1) ...seis(nrows,1) seis(1,2) seis(2,2) ... ...seis(nrows,ncols)]`. Thus the last sample of column 1 and the first sample of column 2 are contiguous in memory. This means that the first sample of column 2 can be addressed either by `seis(1,2)` or by `seis(nrows+1)`. In accordance with this formulation, MATLAB allows any matrix to be entered into a formula as an equivalent column vector using the notation `seis(:)`.

1.6.4 Special Values: NaN, Inf, and eps

MATLAB supports the IEEE representations of NaN (not-a-number) and Inf (infinity). Certain mathematical operations can cause these to arise, and they then behave in defined ways. For example, `0/0`, `Inf*0`, `Inf±Inf`, and `Inf/Inf` all result in NaN. Furthermore, any arithmetic operator involving a NaN is required to produce NaN as an output. This can cause unexpected results. For example, `max([1 2 3 NaN 4 5 6])` results in NaN, and the functions `min`, `mean`, `std`, and many others will also return NaN on this vector. Even worse, running `fft` on data with only a single NaN in it will produce NaN for the entire output spectrum. Despite these apparent complications, NaNs have enough uses that they are well worth having around. The occurrence of a NaN in your output data signals to you that an unexpected arithmetic exception has taken place in your code. This must be found and corrected. NaNs can also be used as convenient placeholders to mark some special point in a matrix because they can easily be found. Also, when a NaN occurs in a graphic, the action is to blank that part of the display. Thus NaNs can be used to temporarily turn off parts of a graphic, something that is especially useful in 3D.

When used in logical comparisons, such as `NaN==5`, the result is almost always False. This means that an expression such as `find(v==NaN)` will not correctly identify any NaNs in the vector `v`. (Of course, it works just fine if you want to find a number, such as in `find(v==5)`.) So, NaNs must be located using `find(isnan(v))`, which returns a vector of the indices of any NaNs in `v`. (The function `isnan` returns a logical vector of 1's and 0's, indicating vector elements that are NaNs.)

Infinity is handled similarly to NaNs in that certain operations generate an infinity that subsequently behaves in a defined way. For example, `x/0` generates an Inf and `x/Inf` always returns zero (unless `x` is Inf, when NaN results). Inf also must be handled specially in logical comparisons. The functions `isinf` and `isfinite` can be used by themselves or as input to `find`, as discussed above with `isnan`. Also, the debug command `dbstop`

`if naninf` is helpful for stopping execution at the point where NaN or Inf occurs in any function.

MATLAB uses double precision floating-point arithmetic for its calculations. This is actually a higher standard of precision than has traditionally been used in seismic data processing. Most processing systems have been written in Fortran or C and use single precision arithmetic. MATLAB supplies the internal constant `eps`, whose value is the smallest positive floating-point number available on your machine. In a loose sense, `eps` represents the machine "noise level" and is about 2.2204×10^{-16} on the first author's computer. (How many decibels down from unity is `eps`?) As a general rule, tests for the equality of two floating-point variables should be avoided. Instead of a test like `if(x==y)`, consider something like `if(abs(x-y)<10*eps)`. `eps` can also be useful in computing certain limits that involve a division by zero. For example, computing a sinc function with `x=0:.1:pi;y=sin(x)./x;` generates a zero divide and `y(1)` is NaN. However, `x=0:.1:pi;y=sin(x+eps)./(x+eps);` avoids the zero divide and results in a `y(1)` of 1.

1.7 Chapter Summary

The scope of this book was defined as a survey of the fundamental numerical methodologies used in seismic exploration, and their physical basis. Reasons for the choice of the MATLAB language for the presentation of algorithms were given. MATLAB was suggested as a superior environment owing to its vector language, extensive numerical toolbox, interactive environment, and built-in graphics. MATLAB conventions for naming functions were explained and the use of numbered code snippets in a theorem-like environment for code presentation was discussed.

Considerable time was spent discussing the display of seismic data. It was argued that the numerical range (dynamic range) of modern data is very large, and this leads to the likelihood of clipping in display. Wiggle trace, WTVA, and image plots were discussed as display tools, and their relative merits were contrasted.

Finally, a brief survey of MATLAB programming was presented, with an emphasis on topics of importance when dealing with large datasets. Scripts and functions were discussed and their structure outlined. Common programming blunders that lead to reduced performance were discussed. Vector addressing, vector programming, and the colon operator were discussed, with an emphasis on how they relate to the creation of efficient code.

2 Signal Theory: Continuous

The theory of digital signal processing is a vast subject and its complete description requires much more space than the chapters available here. As an acquaintance with the fundamentals of the theory is so essential, however, this and the following chapter are included as an introduction. The scope of the present discussion is limited to fundamentals that have direct relevance to later chapters. Those requiring a more thorough exposition should consult a textbook such as Karl (1989).

Concepts will be developed here for both continuous-time signals (Chapter 2) and sampled, discrete-time signals (Chapter 3). The continuous-time viewpoint arises naturally when developing models for physical processes that evolve in real time, often represented as solutions to differential equations with time as a continuous variable. Discrete-time signals arise when solving such equations in complex situations or in processing real seismic data, where digital computations in a computer are the only practical approach. Therefore, both approaches will be explored where appropriate, and the transformation of a continuous expression into a discrete one and the reverse will be demonstrated.

2.1 What is a Signal?

A signal is a stream of data that one can record from any physical measurement device, such as a geophone, microphone, photocell array, or similar device. We represent a signal mathematically as a function $s(t)$, where t is, say, a time variable, and $s(t)$ is the value of the physical state being measured at time t. For instance, a geophone senses the motion of the Earth by coupling this motion to a magnet and a coil of wire, generating a voltage proportional to the velocity of the motion. In this case, the signal $s(t)$ represents the voltage s (in units of volts) at time t (in seconds) generated by the electrical coil in the geophone. Similarly, an acoustic microphone generates a signal $s(t)$ that represents the minute change in air pressure at time t in response to a sound wave impinging on the microphone.

Signals may also be multidimensional, represented as a function of several variables such as $s(x, y)$ or $s(x, y, z, t)$, where x, y, z are variables in space. For instance, an image falling on a photographic plate will create light and dark spots on the plate. There will be an intensity of light $s(x, y)$ at each point (x, y) on the plate, say with s measured in units of lumens and the position (x, y) measured in millimeters. For most of this chapter, we will limit ourselves to one-dimensional signals represented as functions such as $s(t)$, taking values which may be real or complex numbers.

The vast range of physical signals that could occur in nature means that a diverse family of mathematical functions is needed to accurately represent them. So, we do not restrict ourselves only to smooth, continuous functions, and instead allow for more "messy" functions, including stepwise jumps, rapid oscillations, and perhaps even spikes and other singularities. We place only enough restrictions on them such that some reasonable analysis or signal processing can be performed. For one-dimensional signals, a convenient mathematical family is the set of complex-valued[1] functions $s(t)$ which can be approximated by piecewise continuous functions on finite intervals of time. We add the restriction that the signal has finite energy, namely that its square integrates to a finite number:

$$\int_{-\infty}^{\infty} |s(t)|^2 \, dt < \infty. \tag{2.1}$$

The set of all such signals is a vector space: one can add and subtract any two signals, multiply by any constant, and obtain a new signal still with finite energy. These functions can be integrated on any finite interval, and operations such as the Fourier transform will be properly defined for them. Similarly, the time-shifted version of any such signal $s(t + \tau)$ also has finite energy. The finite-energy condition ensures that the time-lagged crosscorrelation between two signals always yields a finite number. That is, the integral

$$(r \otimes s)(\tau) = \int_{-\infty}^{\infty} r(t) s^*(t + \tau) \, dt < \infty \tag{2.2}$$

is finite. Note here that s^* is the complex conjugate of s, and τ is the time lag. The fundamental operations of crosscorrelation and convolution are the basis for much of signal processing.

2.2 Crosscorrelation and Autocorrelation

It is often desirable to compare two signals and compute a numerical measure of their similarity. As in the case of an echo, or a reflected seismic signal, it often occurs that one signal is basically a time-delayed version of another. Therefore, it is useful to measure the similarity of two signals over a range of time shifts. These time shifts of one signal relative to the other are called *lags* and the numerical measure of similarity at different lags is called the *crosscorrelation* of the signals. A special case of crosscorrelation occurs when the two signals are identical, and this is called the *autocorrelation*.

Consider the case of two signals $r(t)$ and $s(t)$ with finite energy. The integral $cc(0) = \int_{-\infty}^{\infty} r(t) s^*(t) \, dt$ is called the crosscorrelation at zero lag and measures the similarity of the two signals. Suppose the two signals are identical; then $r(t)s^*(t) = |r(t)|^2 \geq 0$ will never be negative and $cc(0)$ will have a large positive value compared with any other case when

[1] Most physical devices such as geophones and cameras measure real-valued signals, but there are important cases where complex-valued signals occur, and we retain that possibility here.

the signs of $r(t)$ and $s(t)$ have no relationship. Now suppose $r(t)$ is the negative of $s(t)$.[2] Then $r(t)s(t) = -|r(t)|^2$ will never be positive and $cc(0)$ will be a large negative number. So, large positive values of $cc(0)$ indicate that the signals are similar and in phase, while large negative values indicate similarity but a large relative phase rotation.

For continuous signals, the general crosscorrelation is defined as

$$cc(\tau) = (r \otimes s)(\tau) = \int_{-\infty}^{\infty} r(t)s^*(t+\tau)\,dt, \tag{2.3}$$

where τ is called the *lag time* and s^* is the complex conjugate of s. The order in which the signals r, s appear in the product $(r \otimes s)$ is important, even for real-valued signals, for if the order is reversed a change of variables $t \mapsto t - \tau$ shows that

$$(s \otimes r)(\tau) = \int_{-\infty}^{\infty} s(t)r^*(t+\tau)\,dt = \int_{-\infty}^{\infty} s(t-\tau)r^*(t)\,dt = (r \otimes s)^*(-\tau). \tag{2.4}$$

Which is to say, commuting the signals being crosscorrelated causes the crosscorrelation to reverse in time lag, as well as introducing a complex conjugation.[3]

We will see in the next section that crosscorrelation is related to convolution, and the frequency spectrum of a crosscorrelation can be deduced from the convolution theorem. Let $cc(\tau) = (r \otimes s)(\tau)$; then its spectrum is

$$\hat{cc}(f) = \hat{r}^*(f)\hat{s}(f) = RSe^{\phi_s - \phi_r}, \tag{2.5}$$

where we have used Eq. (2.58), and $\hat{r} = Re^{i\phi_r}$ and $\hat{s} = Se^{i\phi_s}$. Thus the crosscorrelation has an amplitude spectrum given as the product of the amplitude spectra of the two signals, but its phase spectrum is the phase difference of the two signals.

As was mentioned, the autocorrelation is the special case of the crosscorrelation of a signal with itself. For the continuous case, this is

$$ac(\tau) = (s \otimes s)(\tau) = \int_{-\infty}^{\infty} s(t)s^*(t+\tau)\,dt. \tag{2.6}$$

It follows from Eq. (2.5) that when a signal s is real-valued, the autocorrelation is zero phase and hence symmetric about $t = 0$ in the time domain. The zero-lag autocorrelation is $ac(0) = \int_{-\infty}^{\infty} |s(t)|^2\,dt = E$, which is the total trace energy, and the Fourier transform of the autocorrelation is the *energy spectrum*, or the amplitude spectrum squared. The zero-lag value of the autocorrelation is always the largest value with respect to time. Statistical arguments lead to the intuitively plausible result that the autocorrelation of a random time series is $E\delta(\tau)$, where E is the power of the random signal. Even a completely random signal always correlates strongly with itself at zero lag; however, any other lag of any size will destroy the correlation. In fact, the "randomness" of a signal is often assessed by comparing its autocorrelation with a delta function.

[2] This is called a phase rotation of 180° or π radians, since $e^{\pi i} = -1$.

[3] For real signals, there is no complex conjugation, but the time lag still reverses, i.e., $\tau \mapsto -\tau$.

2.2.1 Application of Crosscorrelation

The primary use of crosscorrelation in seismic data processing is in methods of trace alignment. For example, *automatic statics* techniques are methods that often use crosscorrelations to determine the relative time shifts between pairs of traces. For each trace pair, the crosscorrelation is computed and the maximum and its lag time are found. Depending upon the application, either the actual maximum or the maximum absolute value may be preferred. In this context, it is important to find an interpolated maximum, meaning that allowance must be made for the maximum to fall between two samples of the crosscorrelation function. This can be done by finding the maximum sample of the crosscorrelation function and performing a band-limited interpolation to perhaps a tenth of the original sample size between the maximum sample and its nearest neighbors. Then the interpolated maximum is just the maximum interpolated sample. The lag at which this interpolated maximum occurs is an estimate, accurate to $\Delta t / 10$, of the time shift between the traces. `maxcorr` is available for this task. Another use of crosscorrelation is measuring the quality of data processing by calculating crosscorrelations with respect to a known answer such as a seismogram computed from well control.

The major use of autocorrelation is in deconvolution. In an important theoretical result, it can be shown that the inverse of a minimum-phase wavelet is determined entirely by its autocorrelation. Thus, a seismic deconvolution operator can be designed if the autocorrelation of the seismic wavelet can somehow be estimated. This means that only an estimate of the embedded wavelet's power (amplitude spectrum squared) spectrum is needed, which is a much easier thing to estimate than the wavelet's phase.

As an illustration of the computation of crosscorrelation functions, consider Figures 2.1a, 2.1b, and 2.2a. In the first two figures the comparison of two signals, s_1 and s_2, is illustrated for three different lags and for four scenarios: (1) s_2 is identical to s_1, (2) s_2 is a 45° phase rotation of s_1, (3) s_2 is a copy of s_1 but time shifted by $\delta t = 0.01$ s, and (4) s_2 is both time shifted and phase rotated relative to s_1. The three different lags examined are $\tau = [-0.01, 0, 0.01]$ s, so that the maximum should appear at a lag of either $\tau = 0$ or $\tau = 0.01$ s. (There is great possibility of sign confusion here about whether the maximum should be at lag $\tau = 0.01$ s or at $\tau = -0.01$ s. This example has been arranged for the former; however, there is no consistent sign convention.) In each panel of Figures 2.1a and 2.1b, the two traces are shown plotted together with s_2 shifted properly for the lag time. The product $s_{1;k}s_{2;k+j}$ is a pointwise product between time-aligned samples on the two signals. This is exactly what is accomplished by MATLAB's .∗ operator. The crosscorrelation coefficient is then the sum of this pointwise product. When the signals are aligned and in phase, a large positive number near 1.0 is calculated. When they are aligned but a phase rotation exists, the crosscorrelation is lower and will eventually reach -1 for a 180° phase rotation. A crosscorrelation value of zero indicates that there is no similarity between the two signals. In theory, a random time series has zero correlation with all other signals at all lags. When s_2 is a time-shifted copy of s_1, the maximum correlation occurs at a lag corresponding to that time shift. These figures illustrate the computation of only three lags, but for a signal of length N there are $2N - 1$ possible lags. The entire crosscorrelation functions for these four scenarios are shown in Figure 2.2a. In a practical application such

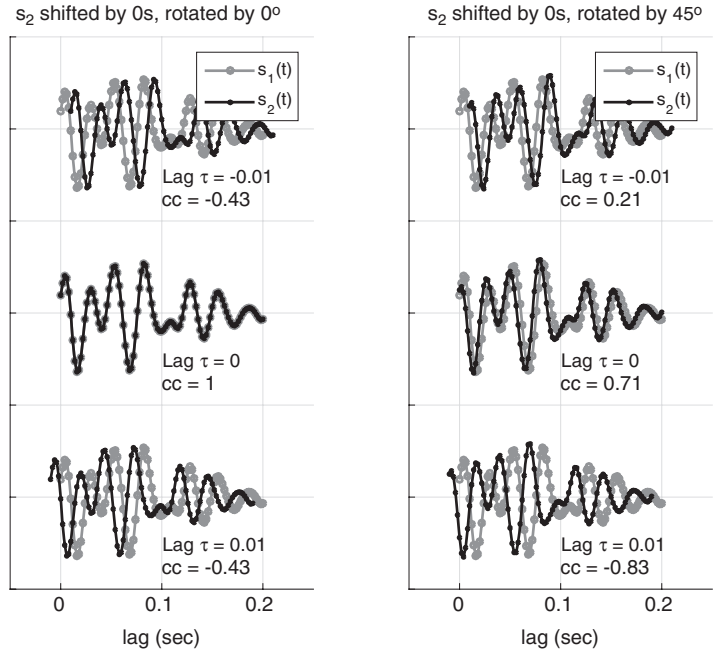


Figure 2.1a The crosscorrelation of two signals s_1 and s_2 is illustrated for lags of $\tau = [-0.01, 0, 0.01]$. In the left panel the two signals are identical, while in the right panel s_2 has a 45° phase rotation. For each lag, s_2 is placed in position relative to s_1 and the product $s_{1;k}s_{2;k+j}$ is formed and then summed. The resulting crosscorrelation values are annotated on the figure, and the entire crosscorrelation function (normalized) is shown in Figure 2.2a.

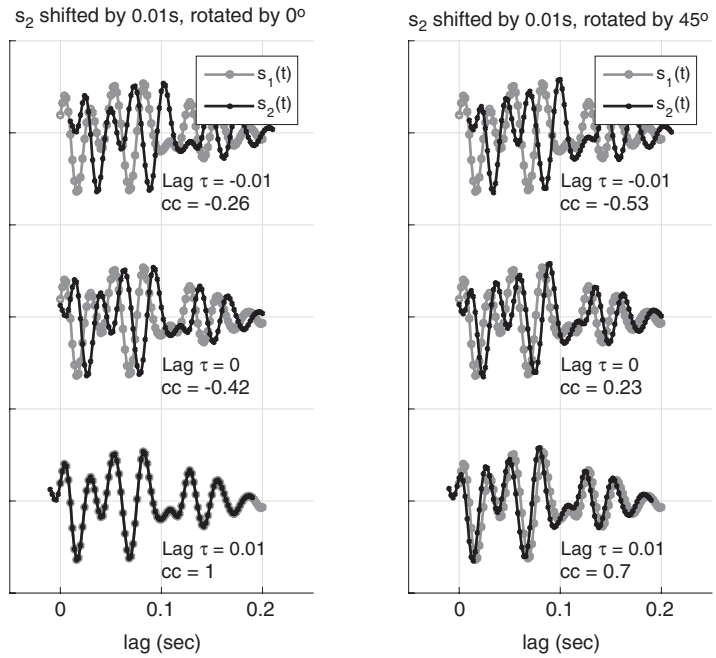


Figure 2.1b Similar to Figure 2.1a except that s_2 has been given a time shift of 0.01 s in both panels. The complete crosscorrelation function is shown in Figure 2.2a.

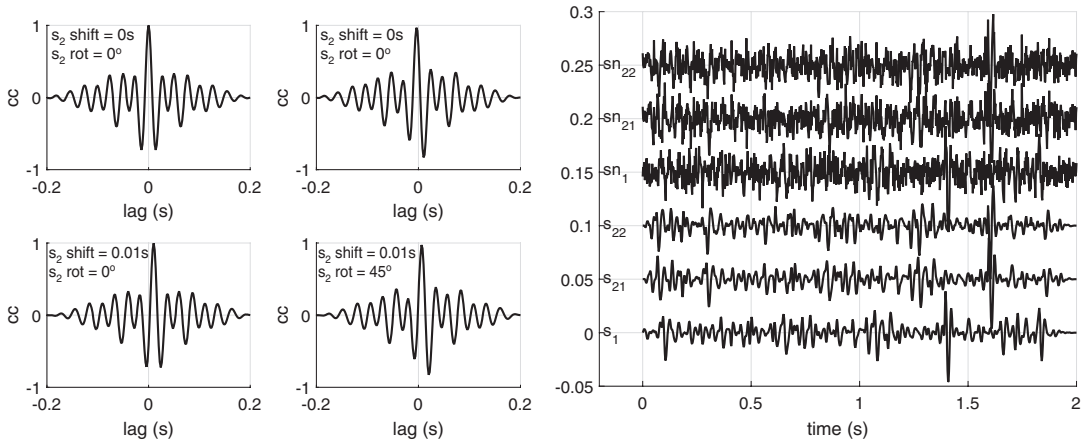


Figure 2.2a (left) The complete normalized crosscorrelation functions for the trace comparisons are illustrated in Figures 2.1a and 2.1b. The top row corresponds to Figure 2.1a, while the bottom row is for Figure 2.1b. A phase rotation of s_2 causes a corresponding phase rotation in the crosscorrelation function. A time shift of s_2 causes the maximum of the crosscorrelation function to appear at a lag corresponding to the time shift.

Figure 2.2b (right) The synthetic seismograms created by Code Snippet 2.2.1: $s_1(t) = (r_1 \bullet w_1)\,(t)$, where w_1 is a 30 Hz dominant minimum-phase wavelet, and $s_{21}(t) = (r_2 \bullet w_1)\,(t)$, where r_2 is similar to r_1 but delayed by 100 samples (0.2 s). The first 100 samples of r_2 are not found on r_1. $s_{22}(t) = (r_2 \bullet w_2)\,(t)$, where w_2 is a 90° phase rotation of w_1. sn_1, sn_{21}, sn_{22} are noisy versions of s_1, s_{21}, s_{22}, respectively, where the signal-to-noise ratio is 1.

as automatic statics, millions of crosscorrelation functions are required and it is common to compute them only out to some maximum lag. This means that relative time shifts greater than this maximum lag will be missed.

As a final example of crosscorrelation, Code Snippet 2.2.1 demonstrates the ability of crosscorrelation to find a large time shift between two signals even in the presence of phase shifts and considerable noise. This simulation creates three different convolutional seismograms for crosscorrelation testing. First, the seismogram s_1 is the convolution of the reflectivity r_1 and a 30 Hz dominant minimum-phase wavelet w_1. Then a reflectivity r_2 is generated that is r_1 delayed by 100 samples (or 0.2 s), where the first 100 samples of r_2 are not found on r_1. Then seismogram s_{21} is the convolution of r_2 with w_1. Also, wavelet w_2 is a 90° phase rotation of w_1 and seismogram s_{22} is the convolution of r_2 and w_2. For each of these three seismograms, noisy copies are also created (sn_1, sn_{21}, and sn_{22}) by adding normally distributed random noise whose power has been scaled to give a signal-to-noise ($s2n$) ratio of 1. These six seismograms are shown in Figure 2.2b. From these seismograms, the four interesting crosscorrelations are $s_1 \otimes s_{21}$, $s_1 \otimes s_{22}$, $sn_1 \otimes sn_{21}$, and $sn_1 \otimes sn_{22}$ and these are shown in Figure 2.3a. All four crosscorrelations detect the 0.2 s time shift although with varying degrees of accuracy. In this figure, the entire crosscorrelation functions are shown for all lags between -0.4 s and 0.4 s as computed by `ccorr`. Also annotated on the figure are the interpolated maxima for each crosscorrelation function and the interpolated lag at which they occur as computed by `maxcorr`. Here `maxcorr` has been instructed to find the maximum positive value of each crosscorrelation function. The

Code Snippet 2.2.1 This code produces the results displayed in Figures 2.2b and 2.3a. The concept is to create a reference seismogram, $s_1(t) = (r_1 \bullet w_1)(t)$, and two comparison seismograms, $s_{21}(t) = (r_2 \bullet w_1)(t)$ and $s_{22}(t) = (r_2 \bullet w_2)(t)$, to be crosscorrelated with s_1. Here the reflectivity r_2 is similar to r_1 but delayed by 100 samples (lines 5–9), w_1 is a 30 Hz dominant minimum-phase wavelet (line 11), and the wavelet w_2 is a 90° phase rotation of w_1 (line 11). These three seismograms are created first without noise (line 13) and then with an *s2n* (signal-to-noise) ratio of 1.0 (lines 14–17). The crosscorrelation functions and the interpolated maxima are then computed for $s_1 \otimes s_{21}$ and $s_1 \otimes s_{22}$ both with and without noise.

```
1
2   dt=.002;tmax=2;%time sample rate and record length
3   fdom=30;s2n=1;%dominant frequency and signal-to-noise ratio
4   phase=90;%phase rotation
5   nlag=100;%delay of the second reflectivity in samples
6   [r1,t]=reflec(tmax,dt,.2,3,5);%first reflectivity
7   r2=[zeros(nlag,1);r1(1:end-nlag)];%r2 is r1 but shifted by nlag
8   rtmp=reflec(2,dt,.2,3,9);%something to fill in the zeros on r2
9   r2(1:nlag)=rtmp(1:nlag);%fill in zeros
10  %wavelets
11  [w1,tw]=wavemin(dt,fdom,.2);w2=phsrot(w1,phase);
12  %signals
13  s1=convm(r1,w1);s21=convm(r2,w1);s22=convm(r2,w2);
14  %make noise
15  n1=rnoise(s1,s2n);n2=rnoise(s1,s2n);
16  %add noise to signals
17  s1n=s1+n1;s21n=s21+n2;s22n=s22+n2;
18  %correlations
19  maxlag=2*nlag;%we will search from -maxlag to maxlag
20  aflag=1;%we will pick positive values for cc
21  cc121=ccorr(s1,s21,maxlag);%the entire correlation function
22  mcc121=maxcorr(s1,s21,maxlag,aflag);%the maximum and its lag
23  cc122=ccorr(s1,s22,maxlag);
24  mcc122=maxcorr(s1,s22,maxlag,aflag);
25  cc121n=ccorr(s1n,s21n,maxlag);
26  mcc121n=maxcorr(s1n,s21n,maxlag,aflag);
27  cc122n=ccorr(s1n,s22n,maxlag);
28  mcc122n=maxcorr(s1n,s22n,maxlag,aflag);
29  tau=dt*(-maxlag:maxlag);%lag vector to plot correlations versus
```

End Code

signalcode / crosscorr_shift_phase_noise .m

best estimates of the time shift come from $s_1 \otimes s_{21}$ and $sn_1 \otimes sn_{21}$, where there is no phase rotation between the wavelets involved. The second of these two crosscorrelations gets the correct lag to within 1 ms (half the sample interval) despite the very noisy seismograms involved. This well-known ability of crosscorrelation to "see through noise" can be understood as a consequence of the summation in the crosscorrelation formula, which reduces random noise. As long as the noises in the two signals are not correlated (i.e., have no

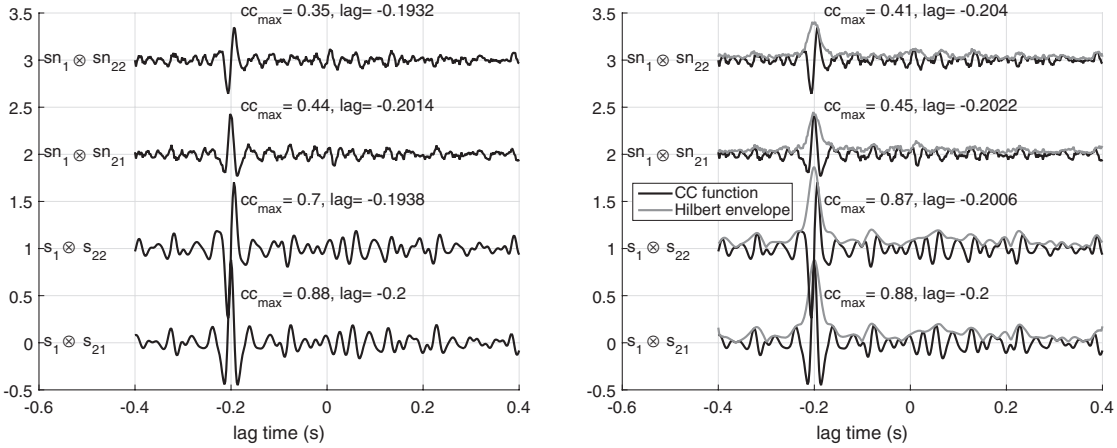


Figure 2.3a (left) The normalized crosscorrelations $s_1 \otimes s_{21}$ and $s_1 \otimes s_{22}$ are shown for the seismograms of Figure 2.2b both with and without noise. Each correlation function has a constant shift upward to allow it to be plotted independently. The values for the maximum and its lag are interpolated from the samples on the crosscorrelation function. The 90° phase shift is apparent in the crosscorrelation functions and this causes that maximum to appear at an incorrect lag. The noisy versions give similar results but with a degradation in the crosscorrelation maximum.

Figure 2.3b (right) A repeat of Figure 2.3a except that the maxima of the crosscorrelation functions are picked from their Hilbert envelopes. The Hilbert envelopes of each crosscorrelation function are also shown.

relationship to each other), then this will happen. The other two correlations show that the maximum correlation is estimated about 6 ms (three times the sample interval) too early, which is a significant error in an autostatics program. This error arises because the phase of the crosscorrelation function is the difference of the phases of the two signals and hence the 90° phase rotation is seen in the main wavelet at lag −0.2 s. Picking the maximum value is making a pick at an incorrect time. In this case the pick should be made at the zero crossing of the 90° wavelet that is seen at −0.2 s. However, if the phase rotation is unknown, as is usually the case, then this is impractical. Instead, `maxcorr` has the option to place the pick at the maximum of the Hilbert envelope. If this is done as shown in Figure 2.3b, then the picked lags become −0.200 s for $s_1 \otimes s_{21}$, −0.201 s for $s_1 \otimes s_{22}$, −0.202 s for $sn_1 \otimes sn_{21}$, and −0.204 s for $sn_1 \otimes sn_{22}$, which is a clearly improved result. The Hilbert envelope works well in this case, but it can lead to problems when the phase differences are not constant (in frequency) or there are phase changes caused by the reflectivity, not the wavelet.

2.3 Convolution

The process of convolution is so fundamental to all of signal theory that its description is an excellent starting point. A variety of descriptions of convolution are given in this section. They are all useful in different contexts and some will appeal more to the intuition than

others. Fundamentally, however, these are all merely different perspectives on the same mathematical expression, which is stated here for continuous-time signals. Suppose signal $a(t)$ is to be convolved with signal $b(t)$ to give a new signal $c(t)$. The mathematical formula for the convolution is

$$c(t) = \int_{-\infty}^{\infty} a(\tau)b(t-\tau)\,d\tau = (a \bullet b)\,(t)\,. \tag{2.7}$$

Comparing this with the formula in Eq. (2.3) for the crosscorrelation, a change of variables $\tau \mapsto -\tau$ yields

$$c(t) = \int_{-\infty}^{\infty} a(-\tau)b(t+\tau)\,d\tau = \left(a' \otimes b\right)(t), \tag{2.8}$$

which shows that the convolution $(a \bullet b)\,(t)$ is just the crosscorrelation of the time-reversed signal $a'(t) = a(-t)$ with the (real) signal $b(t)$. Note that it is conventional to choose τ as the integration variable and t as the independent variable in the convolution integral, which is just the opposite of what is common for the crosscorrelation.

The rightmost expression $a \bullet b$ in Eq. (2.7) is called the *abstract form* of convolution and does not describe how the computation is done. It simply uses the $\bullet$ symbol to denote that convolution is happening, and the parentheses indicate that the result depends on t. Sometimes this is written as $c = a \bullet b$ or as $c(t) = a(t) \bullet b(t)$. The finite-energy condition on our signals ensures this integral is always a finite number. A change of variables $\tau' = t - \tau$ in the integral shows that the order of the signals a, b is immaterial, and thus $a \bullet b = b \bullet a$. We say that convolution is *commutative*, which is certainly not the case for crosscorrelation.

Since the integral in Eq. (2.7) can be approximated by a sum of function values times interval widths, we have

$$c(t) \approx \sum_k a(\tau_k)b(t-\tau_k)\Delta_k, \tag{2.9}$$

which indicates that the convolution is approximately a weighted sum of translates $b(t-\tau_k)$ of the second signal $b(t)$. For instance, if the function $a(t)$ is a simple spike[4] that is only nonzero close to a point τ_0, then the convolution gives

$$c(t) \approx a(\tau_0)b(t-\tau_0)\Delta_0, \tag{2.10}$$

showing that the result is approximately the original signal b shifted in time by τ_0, and weighted by the factor $a(\tau_0)\Delta_0$.

Note that since convolution is commutative, it is also true that the convolution is approximately a weighted sum of translates of the first signal $a(t)$.

The integral in Eq. (2.7) explicitly defines convolution, and for someone proficient in calculus, this definition is sufficient. Examination shows that the two functions being convolved are multiplied under the integral sign, with a somewhat complicated functional

[4] A spike is a boxcar function concentrated on a narrow interval. The product of its width and height is called the weight of the spike.

dependence for each. Since the integral is a limit of a sum of products, the formula indicates that convolution has some properties of both multiplication and addition. In the next section, an intuitive perspective on convolution will be developed. This is presented as fact without justification or linkage with Eq. (2.7). The connection between concept and equation will emerge in later sections.

2.3.1 Convolution by Intuition

An understanding of convolution is enhanced by developing a strong intuition for the process. Possibly the most common use of convolution in seismic analysis is the *convolutional model*, which expresses the creation of a synthetic seismogram, or trace, by the convolution of a *wavelet* with a *reflectivity*, or $s(t) = (r \bullet w)\,(t)$. Here, "wavelet," or w, refers to a characteristic function of time that represents a *source waveform*, or the signal emitted by a source. The other function, "reflectivity," or r, is a representation of the Earth's response to an idealized impulse injected down into the Earth at some surface location. The reflectivity is usually expressed as a function of time, where the function's value at time t is the normal-incidence P-wave reflection coefficient for a point, vertically beneath the source location, whose two-way traveltime (i.e., the traveltime from source to reflector and back to the source) is t. Thus we are imagining a source and a receiver at the same location, in what is called a *zero-offset* experiment. Reflectivity functions are routinely constructed from well-logging measurements made in boreholes. Here we have explicitly chosen the P-wave reflectivity, meaning that we consider incident pressure waves reflecting as pressure waves. We could consider other types as well, such as S-wave reflectivity (shear waves reflecting as shear waves) or even P–S reflectivity, but the latter would require considering nonzero offsets.

Figure 2.4a shows two typical wavelets, called w_{z} and w_{m}, where the subscripts refer to the common jargon *zero phase* and *minimum phase*. For now these are simply names, but relevant to the current discussion are the observations that the zero-phase wavelet is symmetric about the point $t = 0$, while the minimum-phase wavelet is nonzero only for $t \geq 0$. The latter is called *causal*, while the former is *noncausal*. Both wavelet types are frequently encountered in seismology, but real seismic sources can emit only causal wavelets. (We always think of $t = 0$ as the instant of source initiation.) The noncausal wavelets arise in seismic processing, especially after deconvolution and well-tying. A real seismic source, such as a Vibroseis vehicle, can emit a wavelet that looks much like w_{z} but there must be a time delay such that the entire wavelet exists only for $t \geq 0$. In the lower part of Figure 2.4a is shown the result of convolving these two wavelets with a very simple reflectivity function. In this case, $r(t)$ is nonzero only near times $t_1 = 0.350\,\mathrm{s}$ and $t_2 = 0.650\,\mathrm{s}$, where its spikes achieve the weights $r_1 = +0.1$ and $r_2 = -0.2$, respectively. The characteristic behavior of $w \bullet r$ is that each reflection spike on r is replaced by a scaled copy of the wavelet. The $t = 0$ sample of the wavelet is placed directly on the reflection spike and the wavelet is scaled by the weight of the spike. Thus the two convolutions shown each have two copies of the wavelet, where the second wavelet copy is inverted in polarity and twice as large as the first. Physically, this says that a 1D Earth with two reflectors of magnitude

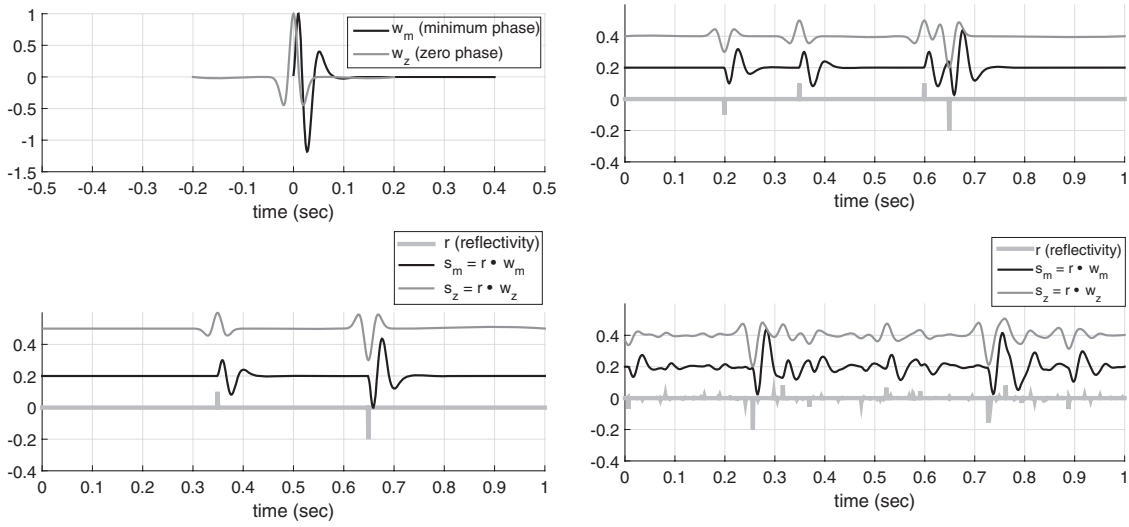


Figure 2.4a (left) Top: Two wavelets, one zero phase (noncausal) and the other minimum phase (causal). Both are normalized to 1.0. Bottom: The result of convolving each wavelet with a very simple two-spike reflectivity is shown below the wavelets. The convolved results are shifted upward by 0.2 and 0.4 to avoid overplotting. Convolution replaces each reflectivity spike with a copy of the wavelet. The wavelet copies are scaled by the weight of the spike, and the $t = 0$ sample of the wavelet is aligned with the spike. A negative reflection coefficient causes a reversed-polarity wavelet.

Figure 2.4b (right) The two wavelets of Figure 2.4a are convolved with more complicated reflectivities. Top: Two additional reflection spikes are included. The spike at $t = 0.2$ s is sufficiently isolated from the spike at $t = 0.35$ that their wavelets do not interfere. However, there is interference between the spikes at $t = 0.6$ s and $t = 0.65$ s. Bottom: Now the reflectivity has nonzero reflection coefficients everywhere although a few are much larger than the average. There is interference everywhere, and it is impossible to recognize individual wavelets with certainty.

r_1 at time t_1 and r_2 at time t_2 has a seismic response that shows two reflection events at times t_1 and t_2, where the reflection events are described by $r_1 w(t-t_1)$ and $r_2 w(t-t_2)$, and where w refers to either wavelet. (This seismic trace model does not account for multiple reflections.) The meaning of $w(t - t_1)$ is that it represents the wavelet shifted in time to place its $t = 0$ sample at $t = t_1$.

Figure 2.4b extends this example to more complicated reflectivities. We can visualize the computation exactly as before, where each reflectivity spike is replaced by a scaled copy of the wavelet. After this scaling and replacement is done for each reflectivity sample, the resulting wavelets are summed together in a process called *superposition*. In the upper panel of Figure 2.4b, the reflectivity has two new spikes compared with the previous example. The spike at $t_1 = 0.2$ s is sufficiently separated from that at $t_2 = 0.35$ s that the resulting wavelets do not influence one another when superimposed. In this case we say that there is no *interference*. In contrast, the spikes at $t_3 = 0.6$ s and $t_4 = 0.65$ s are separated by only 0.05 s and the wavelets are 0.1 s long, so they do interfere when superimposed. Thus at t_1 and t_2 we see clearly identifiable copies of the wavelets, although with

different scale factors, while for times near t_3 and t_4 we see a more complicated waveform with no clearly identifiable wavelet.

In the bottom panel of Figure 2.4b, we see a much more realistic case. Now the reflectivity is nonzero everywhere (it was generated by the `reflec` command, which uses a random number generator), although there are only a few large spikes and most spikes are very small. The convolution $w \bullet r$ is computed exactly as before, by replacing each reflectivity sample with a scaled copy of the wavelet that has been shifted to the location of the reflectivity sample. When these scaled copies are superimposed (added up), the results are the complicated-looking synthetic seismograms shown. With prior knowledge of the wavelet in each case, it is possible to discern nearly undistorted copies of the wavelet near times 0.25, 0.55, and 0.72 s, but in general the result is very complex. If the wavelet is considered unknown, we cannot confidently identify it anywhere. This is the case in real-world seismic data processing, where it is a fundamental goal to take signals for which $s = w \bullet r$ is a good model and uncover or estimate the underlying reflectivity. This is a very challenging task when the wavelet is unknown, and even more so with the additional complication of additive noise (i.e., $s = w \bullet r + n$, where n is random noise). This topic is called *deconvolution* and is the subject of ongoing research. We will discuss it in much more detail in Chapter 4.

2.3.2 Convolution as Filtering

Convolution is the name given to the process of applying a *stationary* linear filter to a signal. In the context of the previous section, a wavelet is a type of filter, and a reflectivity is a type of signal. By stationary, it is meant that the filter response to an input impulse is independent of the time of the impulse. Another name for this property of stationarity is *translation invariance*. In the previous section, stationarity was manifest in that each reflectivity sample was replaced by a scaled copy of the same fundamental wavelet. The filter is linear because the response to an input $a_1(t) + a_2(t)$ is the sum of the responses to $a_1(t)$ and $a_2(t)$ taken separately. Such a linear, stationary filter is completely specified by its impulse response, $b(t)$. This is defined as the output of the filter when the input is a unit impulse at $t = 0$. Often the phrase *the filter* will be used as a synonym for *the impulse response of the filter*. Suppose the input to the filter is an impulse of magnitude a_1 at time t_1 and a second impulse of magnitude a_2 at time t_2. Then the filter output is

$$c(t) = a_1 b(t - t_1) + a_2 b(t - t_2). \tag{2.11}$$

The property of linearity has been used to form the scaled sum of two time-shifted copies of the impulse response. These appear in the equation as $b(t-t_k)$, where $k = 1$ or $k = 2$. Thus $b(t = 0)$ appears in $c(t)$ at both $t = t_1$ and $t = t_2$. Filters that represent physical systems have the *causality* property that $b(t) = 0$ for $t < 0$. If $b(t)$ is causal, then Eq. (2.11) places the first value that is possibly nonzero at the times of the input impulses. The property of stationarity has been used in that the response to the input impulse at $t = t_2$ is the same function $b(t)$ as for the impulse at $t = t_1$.

Suppose the input to the filter is an arbitrary sequence of N impulses defined by $[a_k] = [a_0, a_1, \ldots, a_{N-1}]$ and the corresponding times $[t_k]$. Then Eq. (2.11) generalizes to the one-dimensional convolution sum

$$c(t) = \sum_{k=0}^{N-1} a_k b(t - t_k). \tag{2.12}$$

Both this equation and Eq. (2.11) are examples of convolution equations, though even Eq. (2.12) assumes too much about the nature of the input to be considered general. The defining feature is the formation of a superposition of scaled-and-delayed copies of a single function, $b(t)$. If a completely general expression is desired, then it must accommodate an $[a_k]$ that is a function of continuous time rather than a series of impulses. As a first step, we define a function $a(t)$ such that $a(t_k)\Delta_k = a_k$ on the interval $[t_{k-1}, t_k]$ of length Δ_k, and is zero otherwise. Then Eq. (2.12) can be rewritten as

$$c(t) = \sum_{k=0}^{N-1} a(t_k) b(t - t_k) \Delta_k. \tag{2.13}$$

In a limiting procedure, we allow the set of times $[t_k]$ at which there are nonzero impulses to extend to all possible times, and Eq. (2.13) approaches

$$c(t) = \int_{-\infty}^{\infty} a(\tau) b(t - \tau)\, d\tau. \tag{2.14}$$

Here, in comparison with Eq. (2.13), an integration has replaced the summation, the integration variable τ has taken the place of the summation index k, and the signal of impulses $[a_k]$ has become the function $a(\tau)$. The presence of the time differential dt in Eq. (2.14) matches the factor Δ_t in Eq. (2.13), making the formulas dimensionally consistent. The limits of integration are formally written from $-\infty$ to ∞ to accommodate all possible input signals. If $a(\tau)$ is causal, then the lower limit may be altered to zero. If $a(\tau)$ vanishes for $\tau > \tau_{\max}$, then the upper limit may be set to $\tau_{\max}$. In other words, the integration extends over all relevant values of τ.

Convolution arises in many parts of physical theory other than the application of linear, stationary filters. In all cases, the interpretation given here of forming a superposition of scaled and time-shifted copies of a physical response (or impulse response) is appropriate. However, the choice of which function, a or b, to interpret as the impulse response is arbitrary. Since convolution is commutative, i.e., $a \bullet b = b \bullet a$, then either function can be interpreted as the impulse response, with the other being the input signal that provides the weighting. This means that all signals can be considered as filters and all filters as signals.

In closing this section, it is appropriate to comment on the meaning of the word *filtering* in this context. In ordinary life, filtering is encountered in situations where something, for example water, is separated into desirable and undesirable parts. Usually, water is passed through a physical device called a filter that somehow traps and removes impurities, thereby separating the water into pure water and contaminants. It is probably not obvious at this point that convolution has a similar action on signals. The connection will

become clear when the link between convolution and the Fourier transform is explicitly made later in this chapter. The Fourier transform is a tool that can decompose a signal into a continuous sum of sines and cosines at various frequencies. What distinguishes one signal from another are the amplitudes (i.e., strengths) and phases (i.e., starting values) of each sinusoid at these various frequencies. The list of amplitudes and phases as a function of frequency forms another signal, called the Fourier transform or Fourier dual, of the time-domain signal. Every distinct signal has a unique Fourier transform that describes which frequencies have strength in the signal. When two signals are convolved, their Fourier dual signals are multiplied, and this is a filtering operation in the frequency domain that is a type of selective separation. More explicitly, special signals called band-pass filters are commonly designed that have strength in the Fourier domain over a chosen set of frequencies called the passband. When such a filter is convolved with a data signal, the result is a filtered version of the data that emphasizes those frequencies in the data corresponding to the passband of the filter. A thoughtful choice of the filter passband can suppress noise, reveal hidden signal, or do both. While this is a direct, intentional filtering of a signal, all convolutions are said to have a filtering operation that is defined by the product of their Fourier duals. Our choice to call one of the two functions in a convolution a filter and the other a signal is entirely arbitrary. In fact, all signals are filters and all filters are signals.

Exercises

2.3.1 Show that convolution is distributive. That is, $a \bullet (b+c) = a \bullet b + a \bullet c$. Do this for both the continuous form of Eq. (2.14) and the discrete form of Eq. (3.37). Is it better to stack (sum) a series of traces and then filter the stacked trace or to filter each one and stack the filtered traces? Why?

2.3.2 Let $u(t) = \exp(2\pi i f t)$. For an arbitrary function $v(t)$, use Eq. (2.14) and show that the convolution $u \bullet v$ is equal to $a(f)u$, where $a(f)$ is a function of frequency and of v. Determine the precise form of a to show how it depends on v.

2.3.3 There are several ways to write down an integral that looks like a convolution integral to the untrained eye but is actually wrong. Consider the convolution $c(t) = (a \bullet b)(t)$. In each of the formulas below, the proposed convolution integral is incorrect. Identify the mistakes and correct them.

$$c(t) = \int_{-\infty}^{\infty} a(\tau) b(\tau - t)\, d\tau,$$

$$c(t) = \int_{-\infty}^{\infty} a(t) b(t - \tau)\, d\tau,$$

$$c(t) = \int_{-\infty}^{\infty} a(\tau) b(t - \tau)\, dt,$$

$$c(t) = \int_{-\infty}^{\infty} a(\tau) b(\tau + t)\, d\tau,$$

$$c(t) = \int_{-\infty}^{\infty} a(\tau)b(\tau)\,d\tau,$$
$$c(t) = \int_{-\infty}^{\infty} a(\tau)b(t)\,d\tau.$$

Here are some rules to remember for a correct equation: (1) The functional dependence of the result $c(t)$ – that is, the variable t in this case – should be found under the convolution integral in *only one* of the two functions and should not be the integration variable. (2) If you add the functional dependences of the two functions in the integrand, the variable of integration should cancel and you should get the functional dependence of the result. That is, in Eq. (2.14), you get $\tau + t - \tau = t$. Both of the following correct forms pass these tests:

$$c(t) = \int_{-\infty}^{\infty} a(\tau)b(t-\tau)\,d\tau,$$
$$c(t) = \int_{-\infty}^{\infty} a(t-\tau)b(\tau)\,d\tau.$$

2.4 The Fourier Transform

The Fourier transform is one of the most significant developments in mathematical analysis. Its digital counterpart, the discrete Fourier transform, plays a central role in signal analysis. Jean-Baptiste Joseph Fourier (1786–1830) developed his ideas in the context of solving the heat equation to study heat dissipation while boring out cannons for the French military. Despite this specific intent, it soon became apparent that Fourier's theory could be used to develop solutions to other partial differential equations, such as Laplace's equation and the wave equation. Even with the success of practical applications, the mathematics community resisted Fourier's ideas because of the difficult challenges they posed for analysis. For example, a discontinuous function, such as the boxcar function, could be synthesized from the superposition of an appropriate set of sine and cosines that are everywhere continuous. How could it be possible to form a discontinuous function by an addition of continuous functions? Also problematic was the convergence of Fourier's series of sines and cosines. What conditions are necessary to ensure the convergence of a Fourier series? The resolution of these and other similar questions is a fascinating chapter in the history of mathematics but is beyond the scope of this text (see, e.g., Korner (1988)). For our purposes, we will treat Fourier theory as the established fact that it is today.

2.4.1 The Temporal Fourier Transform and Its Inverse

Fourier's theory provides a means to represent an arbitrary function as a superposition (sum or integral) of a set of simpler functions called *basis functions*. Usually, these basis functions are trigonometric sines and cosines of different frequencies. To construct a specific

function, the sines and cosines must have amplitudes and phases that depend on frequency. Considered as a function of frequency, the amplitudes comprise the *amplitude spectrum* and the phases are the *phase spectrum* of the function being represented.

Fourier's theory can be defined on either a finite interval or an infinite domain. On a finite interval, only a countable number of basis functions are required, while over an infinite domain the number of basis functions is uncountable. Their superposition is a summation over a countable set or an integration over a continuous variable of frequencies, and the methods are called Fourier series or Fourier transforms. The infinite-domain Fourier transform is the tool of choice for theoretical analysis of the wave equation and other partial differential equations. In data processing, signals persist over a finite length and are sampled, and the discrete Fourier transform (DFT) as implemented by the fast Fourier transform algorithm (FFT) is used.

In this chapter, we consider the case of an infinite interval $(-\infty, \infty)$. Here an uncountable number of basis functions are required, and their superposition is accomplished by an integration over frequency. The calculation of the amplitudes and phases of the basis functions is called the forward *Fourier transform* (often the "forward" is dropped), and the reconstruction of the signal by superposition of the basis functions with their proper amplitudes and phases is called the *inverse Fourier transform*. Rather than treating $\sin(2\pi f)$ and $\cos(2\pi f)$ as separate basis functions, it is common to use $\exp(i2\pi f) = e^{i2\pi f}$. Since Euler's identity says $e^{i2\pi f} = \cos(2\pi f) + i\sin(2\pi f)$ and since the real and imaginary parts of an equation are independent, the complex exponential provides a decomposition equivalent to independent sine and cosine decompositions.

The central result of Fourier theory states that given any signal $s(t)$, we can define its Fourier transform $\hat{s}$ as an integral

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t) e^{-2\pi i f t}\, dt, \tag{2.15}$$

where f is the frequency variable (say, in Hz), and recover that same signal via the inverse Fourier transform, given by the formula

$$s(t) = \int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df. \tag{2.16}$$

In these expressions, $e^{-2\pi i f t}$ is called the Fourier *kernel* and $e^{2\pi i t}$ is the conjugate kernel. The forward transform is often called *Fourier analysis* and the inverse transform is called *Fourier synthesis*.

Proving this result is an exercise in taking limits and verifying that certain integrals are finite. Defining a normalized Gaussian with width 2ϵ and unit mass,

$$\delta_\epsilon(t) = \frac{1}{\epsilon} e^{-\pi t^2/\epsilon^2}, \qquad \int_{-\infty}^{\infty} \delta_\epsilon(t)\, dt = 1, \tag{2.17}$$

it is easy to verify that the convolution of δ_ϵ with a signal s gives

$$(\delta_\epsilon \bullet s)(t) = \int_{-\infty}^{\infty} \frac{e^{-\pi \tau^2/\epsilon^2}}{\epsilon} s(t-\tau)\, d\tau, \tag{2.18}$$

which is the weighted average value of s concentrated on the interval $[t-\epsilon, t+\epsilon]$. This tends to the limit value $s(t)$ as ϵ goes to zero, provided the function s is continuous. On the other hand, the Fourier transform of the normalized Gaussian δ_ϵ as defined by Eq. (2.15) is easily computed to be

$$\hat{\delta}_\epsilon(f) = e^{-\pi\epsilon^2 f^2}, \tag{2.19}$$

which is a (nonnormalized) Gaussian function, tending to one uniformly on any finite interval as ϵ tends to zero. By the dominated convergence theorem,[5] for any integrable signal $s(t)$ we have

$$\int_{-\infty}^{\infty} e^{2\pi i f t}\hat{s}(f)\,df = \lim_{\epsilon\to 0}\int_{-\infty}^{\infty} e^{-\pi\epsilon^2 f^2 + 2\pi i f t}\hat{s}(f)\,df, \tag{2.20}$$

since the weighting factor $e^{-\pi\epsilon^2 f^2}$ tends to one everywhere. The integral on the right is really a double integral, as it also contains an integral defining $\hat{s}$. By Fubini's theorem, we can change the order of integration and put the Fourier transform onto the function $e^{-\pi\epsilon^2 f^2 + 2\pi i f t}$, obtaining the function δ_ϵ shifted by t. Thus we have

$$\int_{-\infty}^{\infty} e^{2\pi i f t}\hat{s}(f)\,df = \lim_{\epsilon\to 0}\int_{-\infty}^{\infty} \delta_\epsilon(t-\tau)s(\tau)\,d\tau = \lim_{\epsilon\to 0}(\delta_\epsilon \bullet s)(t) = s(t), \tag{2.21}$$

as desired.

This proof required the signal s to be integrable ($\int_{-\infty}^{\infty}|s(t)|\,dt < \infty$) and continuous in order for the dominated convergence limit and convolution limit to hold. However, the transforms hold more generally for signals that are only square integrable ($\int_{-\infty}^{\infty}|s(t)|^2\,dt < \infty$) and not necessarily continuous.

Another way to derive the result is to use a Dirac delta function, which can be thought of as a limiting function for the Gaussian functions δ_ϵ. An essential tool in Fourier theory is an *orthogonality relation*. For complex exponentials on the infinite interval, this takes the form

$$\int_{-\infty}^{\infty} e^{2\pi i(f-f')t}\,dt = \delta(f-f'), \tag{2.22}$$

where $\delta(x)$ is the Dirac delta function,[6] which is defined by

$$\int_{-\epsilon}^{\epsilon}\delta(x)\,dx = 1, \tag{2.23}$$

where $\epsilon > 0$ is any positive constant, and

$$\delta(x) = 0, \quad \text{for all} \quad x \neq 0. \tag{2.24}$$

From Eq. (2.23), it follows that

$$\int_{-\epsilon}^{\epsilon}\delta(ax)\,dx = \frac{1}{|a|}\int_{-a\epsilon}^{a\epsilon}\delta(x')\,dx' = \frac{1}{|a|}$$

[5] The dominated convergence theorem in mathematical analysis gives conditions to ensure existence of a limit of integrals, when the integrands converge pointwise.

[6] More precisely, the Dirac delta function is a distribution, or generalized function.

and therefore that

$$\delta(ax) = \frac{1}{|a|}\delta(x). \tag{2.25}$$

The fundamental property of the delta function is

$$\int_{-\epsilon}^{\epsilon} s(x)\delta(x)\,dx = s(0), \tag{2.26}$$

where $s(x)$ is any continuous function. This result is called the *sifting property* of the delta function.

Strictly speaking, the Dirac delta is not a proper function but rather a distribution, defining a linear functional characterized by this sifting property. Physicists usually conceive of the Dirac delta as a representation of a unit spike concentrated at a single point called the *singularity*. This special point is where the argument of the delta function vanishes, which in this case is $x = 0$. Notice that Eq. (2.24) says that the delta function is zero everywhere except at this point, but we do not give its value at the singularity. Instead we say in Eq. (2.23) that an integration across any infinitesimal interval containing the singularity always gives unity, no matter how small or large the interval. Since a single point has no width, this leads to the concept that the value of the delta function at the location of the singularity is infinite. Physicists usually have no problem with the thought that the area under the singularity is $0 \times \infty = 1$, but mathematicians often do. The delta function is actually not a function in the classical mathematical sense, because it has vanishing support[7] and takes on a singular value (infinity) where the argument vanishes. It has been given rigorous meaning through the branch of analysis known as distribution theory and can be defined as the limit of a special sequence of ordinary functions. Each function in the sequence must have unit area but as the limit is taken their support vanishes, meaning that they become more and more narrow. For example, a boxcar of width a and height a^{-1} has unit area and converges to a delta function in the limit as $a \to 0$ (Figure 2.5). This "unit area with vanishing-width property" is what creates the sifting action of Eq. (2.26). Multiply any arbitrary function by a Dirac delta function and integrate, and the result is the arbitrary function evaluated at the location of the Dirac singularity.

A proof of Eq. (2.22) that does not assume the Fourier inversion theorem requires distribution theory, and that is beyond the scope of this discussion. Intuitively, it can be understood quite simply. If $f = f'$, then $\exp(2\pi i(f - f')t) = \cos(0) + i\sin(0) = 1$ and the integral in Eq. (2.22) is infinite. On the other hand, if $f \neq f'$, then $\exp(2\pi i(f - f')t) = \cos(2\pi(f - f')t) + i\sin(2\pi(f - f')t)$. Both terms are simply periodic oscillatory functions that, over the domain $-\infty < t < \infty$, will integrate to zero. The thoughtful reader will realize that if f' is very close to f then $\exp(2\pi i(f - f')t)$ will oscillate very slowly. However, the integration from $-\infty$ to ∞ will always contain infinitely many oscillations (unless f exactly equals f') and will give zero. The factor of 2π in the exponential is required to normalize the integral, resulting in exactly one times the Dirac delta.

[7] The support of a function, $u(x)$, is the set of values of x for which $u \neq 0$. In the case of the delta function, this is a single point.

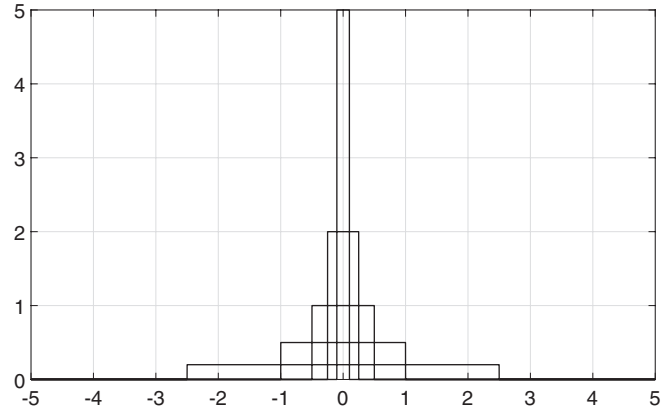


Figure 2.5 Five boxcar functions (rectangular pulses), which all have unit area. The height of each boxcar is the inverse of its width. In the limit as the width shrinks to zero, we obtain the Dirac delta.

Given the orthogonality relation, the Fourier transform theorem can be derived by postulating an expansion of an arbitrary function $s(t)$ in terms of the complex exponentials

$$s(t) = \int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df. \tag{2.27}$$

This says that a representation of $s(t)$ is sought as the superposition of an infinite set of complex exponentials (the basis functions) having complex weights $\hat{s}(f)$. The Fourier transform $\hat{s}(f)$ is sometimes called the Fourier dual of $s(t)$.[8] At this point, $\hat{s}(f)$ is unknown but it can be calculated by multiplying Eq. (2.27) by $\exp(-2\pi i f' t)$ and integrating over t, that is,

$$\int_{-\infty}^{\infty} s(t) e^{-2\pi i f' t}\, dt = \int_{-\infty}^{\infty} e^{-2\pi i f' t} \left[\int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df \right] dt. \tag{2.28}$$

Now, we interchange the order of integration on the right-hand side and use the orthogonality relation, Eq. (2.22), to obtain

$$\int_{-\infty}^{\infty} s(t) e^{-2\pi i f' t}\, dt = \int_{-\infty}^{\infty} \hat{s}(f) \left[\int_{-\infty}^{\infty} e^{-2\pi i (f - f') t}\, dt \right] df \tag{2.29}$$

$$= \int_{-\infty}^{\infty} \hat{s}(f) \delta(f - f')\, df = \hat{s}(f'). \tag{2.30}$$

The term in square brackets in the second expression is the Dirac delta function, and this allows easy evaluation of the df integral. In reaching the final expression, it is often said

[8] In many fields, such as engineering, it is common to denote the Fourier transform by a capital letter like $S(f)$. The notation of the "hat" used here is common in mathematics. Both notations are acceptable. Another common, although unfortunate, practice is to use the same letter for both the function and its Fourier transform and to denote which is which by the dependence on time or frequency. That is, $s(t)$ is the function and $s(f)$ is the Fourier transform. This notation is formally incorrect, since it implies that the function and its Fourier transform are somehow the same function with just different inputs. It should be avoided.

that the Dirac delta function "collapses" the integration to the single point $f = f'$. Upon renaming f' to f, this is just $\hat{s}(f)$, the *Fourier transform* of $s(t)$. In summary, the (forward) Fourier transform of $s(t)$ is given by

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t) e^{-2\pi i f t}\, dt \tag{2.31}$$

and the inverse Fourier transform is

$$s(t) = \int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df. \tag{2.32}$$

As is common in geophysics, the above discussion uses the *cyclical frequency* variable, f, in units of *hertz* or cycles per second. In some instances, it is useful to use the *angular frequency* ω in Eqs. (2.31) and (2.32), as measured in radians per second. Since there are 2π radians in one cycle of a sine wave, these variables are related by $\omega = 2\pi f$. Upon change of variables, the Fourier transform pair in angular frequency is

$$\hat{s}(\omega) = \int_{-\infty}^{\infty} s(t) e^{-i\omega t}\, dt \tag{2.33}$$

and

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{s}(\omega) e^{i\omega t}\, d\omega, \tag{2.34}$$

where the new factor of $1/2\pi$ in the second integral comes from the change of variables with $d\omega = 2\pi\, df$. In addition to the selection of frequency variables, there are other arbitrary choices in the definition of a Fourier transform pair. Things work equally well if the signs in the exponent of the Fourier kernel and its conjugate are reversed: one negative, the other positive. Also, the factor $(2\pi)^{-1}$ in front of the inverse transform (Eq. (2.34)) can be replaced by $(2\pi)^{-1/2}$ in front of both transforms. These relatively arbitrary choices appear in all possible combinations throughout the relevant literature. The practitioner must be aware of this and always check the definitions that are in effect before using a particular theorem or result. Even software like MATLAB must use a particular sign convention for its FFT functions. In this book, the temporal Fourier transforms will always be written as in Eqs. (2.15)–(2.34); however, the choice of angular or cyclical frequency variables will vary as is convenient.

Exercises

2.4.4 Verify that the Gaussian $e^{-\pi t^2}$ is indeed normalized, so that $\int_{-\infty}^{\infty} e^{-\pi t^2}\, dt = 1$. Hint: Use the double integral identities

$$\left(\int_{-\infty}^{\infty} e^{-\pi t^2}\, dt \right)^2 = \int_{-\infty}^{\infty} e^{-\pi x^2}\, dx \int_{-\infty}^{\infty} e^{-\pi y^2}\, dy = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-\pi (x^2 + y^2)}\, dx\, dy \tag{2.35}$$

and convert into polar coordinates, and then integrate with respect to polar variables r, θ.

2.4.5 Show that the Fourier transform of the Gaussian $e^{-\pi t^2}$ is exactly $e^{-\pi f^2}$. Hint: Complete the square of the argument in $e^{-\pi t^2 - 2\pi i f t}$ as

$$-\pi t^2 - 2\pi i f t = -\pi(t - if)^2 - \pi f^2 \tag{2.36}$$

and make the obvious change of variables in the Fourier transform formula (2.31).

2.4.6 Show that the Fourier transform of a perfect unit impulse at $t = 0$ has all frequencies of equal magnitude. This means showing that the Fourier transform of $\delta(t)$ is 1.

2.4.7 Calculate the Fourier transform of $\delta(t - t_0)$.

2.4.8 Show that the convolution of an arbitrary function $u(t)$ with $\delta(t - t_0)$ is $u(t - t_0)$. This means that the convolution with $\delta(t - t_0)$ causes a constant time shift (also called a *static* shift). Make a sketch showing all three functions.

2.4.9 Let $b_a(t)$ be the unit boxcar of width $2a$ defined by $b_a(t) = 1, -a \le t \le a$, and $b_a(t) = 0$ otherwise. Show that the Fourier transform of $b_a(t)$ is $\hat{b}_a(f) = 2a\,\mathrm{sinc}(2af)$ where $\mathrm{sinc}(\theta) = \sin(\pi\theta)/(\pi\theta)$. Make a sketch of both $b_a(t)$ and $\hat{b}_a(f)$. If the width of $\hat{b}_a(f)$ is defined as the distance between the first zero crossings on either side of $f = 0$, show that the product of the widths of $b_a(t)$ and $\hat{b}_a(f)$ is a constant. Determine the constant.

2.4.2 Spectra and Symmetries

For a real- or complex-valued function $s(t)$, the Fourier transform is always complex-valued. It is customary to call $\hat{s}(f)$ either the Fourier transform or the *spectrum* of the function $s(t)$. These terms will be used synonymously in this text. As a complex-valued function, the spectrum has both real and imaginary parts that can be denoted in either of two equivalent ways:

$$\begin{aligned} \hat{s}(f) &= \hat{s}_{\mathrm{R}}(f) + i\hat{s}_{\mathrm{I}}(f), \\ \hat{s}(f) &= \mathrm{Re}\,\hat{s}(f) + i\,\mathrm{Im}\,\hat{s}(f). \end{aligned} \tag{2.37}$$

An alternate decomposition of the spectrum is employed more commonly than that into real and imaginary parts. The *amplitude spectrum*, $A_s(f)$, and the *phase spectrum*, $\phi_s(f)$, are defined, using the polar representation for complex numbers, as

$$\hat{s}(f) = A_s(f)e^{i\phi_s(f)}, \tag{2.38}$$

where $A_s(f) > 0$ is the amplitude of the complex number $\hat{s}(f)$

$$A_s(f) = \sqrt{\hat{s}_{\mathrm{R}}^2(f) + \hat{s}_{\mathrm{I}}^2(f)} \tag{2.39}$$

and $\phi_s(f)$ is the corresponding phase,

$$\phi_s(f) = \arctan\left(\frac{\hat{s}_{\mathrm{I}}(f)}{\hat{s}_{\mathrm{R}}(f)}\right) \text{ or } \phi_s(f) = \mathrm{atan2}(\hat{s}_{\mathrm{I}}(f), \hat{s}_{\mathrm{R}}(f)), \tag{2.40}$$

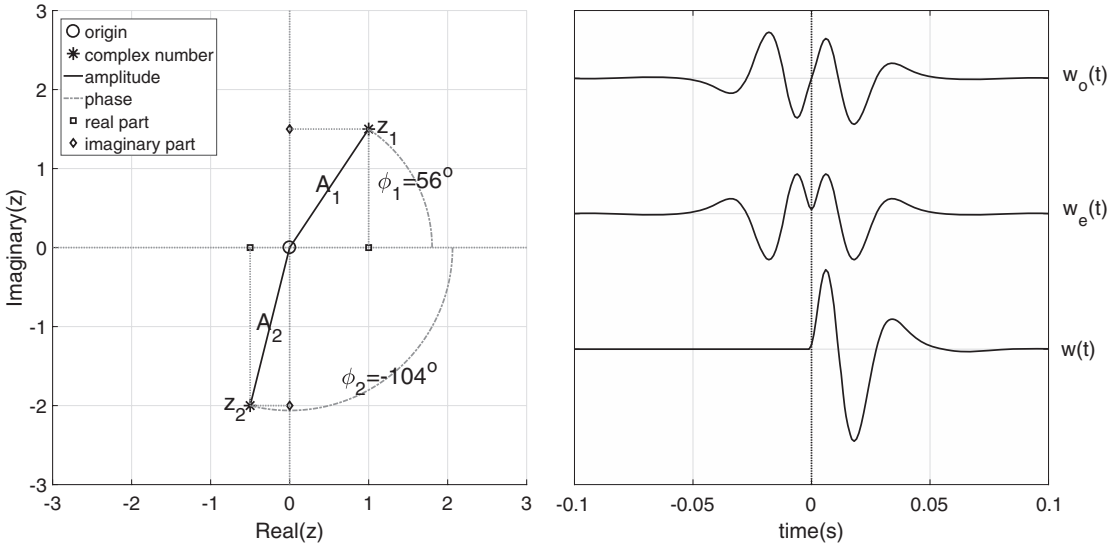


Figure 2.6a (left) Two complex numbers are shown together with their decompositions into amplitude and phase or real and imaginary parts. The numbers are $z_1 = 1 + 1.5i$ and $z_2 = -0.5 - 2i$. Phase angles are measured from the positive real axis. A phase of $-104°$ is equivalent to $360 - 104 = 256°$.

Figure 2.6b (right) A causal wavelet $w(t)$ is shown together with its even and odd parts $w_e(t)$ and odd $w_o(t)$. The vertical dotted line at $t = 0$ is the symmetry marker, and $w_e(t)$ is symmetric about this line while $w_o(t)$ is antisymmetric. The even and odd parts sum to give the asymmetric $w(t)$.

where we use the atan2 function to capture the full circle of possible phases.

Equations (2.37) and (2.38) are equivalent and alternate representations of a spectrum. Either one can be used as convenient for whatever problem is at hand. The spectrum is just a complex-valued function of a real variable f. That is, for every frequency f, the spectrum provides a complex number $z = \hat{s}(f)$ having real and imaginary parts or, equivalently, amplitude and phase. This is depicted graphically for two different complex numbers z_1 and z_2 in Figure 2.6a. The complex numbers of the spectrum of a signal determine the precise amplitude and phase of the basis functions $e^{2\pi ift}$ for every frequency.

The Fourier transform decomposes a function using the complex exponential $e^{2\pi ift} = e^{i\omega t}$ as a basis. By Euler's identity, $e^{i\omega t} = \cos \omega t + i \sin \omega t$. The cosine functions are all *even* functions of t or ω, while the sines are *odd* functions. An even function has the property that $s(-t) = s(t)$, while an odd function has the property that $s(-t) = -s(t)$. Any function can be expressed as a sum of even and odd parts:

$$s(t) = s_e(t) + s_o(t), \tag{2.41}$$

where

$$s_e(t) = \frac{1}{2}\left[s(t) + s(-t)\right] \tag{2.42}$$

and

$$s_o(t) = \frac{1}{2}\left[s(t) - s(-t)\right]. \tag{2.43}$$

Figure 2.6b shows a causal wavelet and its even and odd parts. While the wavelet has no symmetry about $t = 0$, its even and odd parts are symmetric and antisymmetric, respectively, and sum to re-create the wavelet. Even functions have the property that $\int_{-a}^{a} s_e(t)\,dt = 2\int_{0}^{a} s_e(t)\,dt$, while for odd functions $\int_{-a}^{a} s_0(t)\,dt = 0$. In words, when an even function is integrated over an even domain, $-a \le t \le a$, the result is twice the integral over $0 \le t \le a$; however, an odd function always gives zero when so integrated. Furthermore, the product of two odd functions or the product of two even functions is even, while an even function times an odd function is odd.

The Fourier transform can be applied to real-valued or complex-valued signals with equal validity. However, geophysical data is all real-valued, and this fact induces certain symmetries in the Fourier transform. These symmetries are:

- The real part of $\hat{s}(f)$ is an even function of frequency.
- The imaginary part of $\hat{s}(f)$ is an odd function of frequency.
- The amplitude of $\hat{s}(f)$ is an even function of frequency.
- The phase of $\hat{s}(f)$ is an odd function of frequency.
- $\hat{s}(f < 0)$ is determined by $\hat{s}(f > 0)$.

This last item is most important because it means that, while the inverse Fourier transform (e.g., Eq. (2.16)) requires both positive and negative frequencies, it is only necessary to compute and store the positive frequencies (plus $f = 0$), because the negative frequencies can be deduced as needed from symmetry.

To prove the first one, use Euler's identity to rewrite the Fourier transform:

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t) e^{-2\pi i f t}\,dt = \int_{-\infty}^{\infty} s(t)\left(\cos(2\pi f t) - i\sin(2\pi f t)\right)\,dt. \tag{2.44}$$

Then, let $s(t) = s_e(t) + s_o(t)$ and we have

$$\hat{s}(f) = \int_{-\infty}^{\infty} s_e(t)\cos(2\pi f t)\,dt - i\int_{-\infty}^{\infty} s_o(t)\sin(2\pi f t)\,dt, \tag{2.45}$$

where the terms $\int_{-\infty}^{\infty} s_o(t)\cos(2\pi f t)\,dt$ and $\int_{-\infty}^{\infty} s_e(t)\sin(2\pi f t)\,dt$ have been dropped because they are integrations of an odd integrand over an even domain and are zero. From Eq. (2.45), we can identify the real and imaginary parts of $\hat{s}(f)$ as

$$\mathrm{Re}(\hat{s}(f)) = \int_{-\infty}^{\infty} s_e(t)\cos(2\pi f t)\,dt \tag{2.46}$$

and

$$\mathrm{Im}(\hat{s}(f)) = -\int_{-\infty}^{\infty} s_o(t)\sin(2\pi f t)\,dt. \tag{2.47}$$

Examination of Eq. (2.46) shows that all of the frequency dependence lies in $\cos(2\pi f t)$, which means $\mathrm{Re}(\hat{s}(f))$ is even in f because $\cos(2\pi f t)$ is even in f. In similar fashion, Eq. (2.47) shows that $\mathrm{Im}(\hat{s}(f))$ is an odd function of frequency. Now that we have established the symmetries of the real and imaginary parts, it then follows directly from Eqs. (2.39) and (2.40) that the amplitude spectrum is even and the phase spectrum is odd.

The observation in Eq. (2.46) can be generalized. Suppose $u_e(t)$ is any even function of time. Then it follows that $\hat{u}_e(f)$ is purely real (its imaginary part vanishes). In similar fashion, it is not difficult to show that the spectrum of an odd function is purely imaginary.

These symmetries can all be brought together in a single statement that defines how to calculate the negative frequencies from the positive ones. Consider a real signal written as the inverse Fourier transform of its spectrum,

$$s(t) = \int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df. \tag{2.48}$$

Since a real signal is equal to its complex conjugate,[9] then it must be true that

$$\int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df = \left[\int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df\right]^* = \int_{-\infty}^{\infty} \hat{s}^*(f) e^{-2\pi i f t}\, df, \tag{2.49}$$

where $\hat{s}^*$ denotes the complex conjugate of $\hat{s}$. The complex conjugation has changed the right-hand side from an inverse to a forward Fourier transform. Substitution of $f' = -f$ will change it back to an inverse transform. This gives

$$\int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df = \int_{\infty}^{-\infty} \hat{s}^*(-f') e^{2\pi i f' t}\, d(-f') = \int_{-\infty}^{\infty} \hat{s}^*(-f') e^{2\pi i f' t}\, df'. \tag{2.50}$$

In the final integral, f' can be freely renamed to f because in a definite integral the name of the variable of integration is irrelevant. (If this is surprising, consider the fact that $\int_0^4 x\, dx$, $\int_0^4 y\, dy$, and $\int_0^4 \eta\, d\eta$ are all equal to the same number, 8.) So, Eq. (2.50) shows that the spectrum of a real-valued signal has the symmetry

$$\hat{s}(f) = \hat{s}^*(-f)$$

or

$$\hat{s}^*(f) = \hat{s}(-f). \tag{2.51}$$

This means that the negative frequencies are not independent of the positive ones. Rather, one may always be calculated from the other. This is called *Hermitian symmetry* and plays an important role in quantum mechanics.

This symmetry has importance in data processing because it means that only half of the Fourier transform need to be computed, stored, and operated upon. Also, if this symmetry is ignored and both the positive and negative frequencies are calculated and processed, then care must be taken to preserve the symmetry so that, upon inverse transformation, a real-valued signal will result.

Figure 2.7 shows the results of a numerical computation of the spectrum of a 10 Hz causal (minimum-phase) wavelet. The details of this computation will be shown later after

[9] The complex conjugate of a complex number z is written as z^* and is defined as $z^* = \mathrm{Re}(z) - i\,\mathrm{Im}(z)$. The complex conjugate may also be formed by replacing i by $-i$ wherever it occurs in a complex-valued expression.

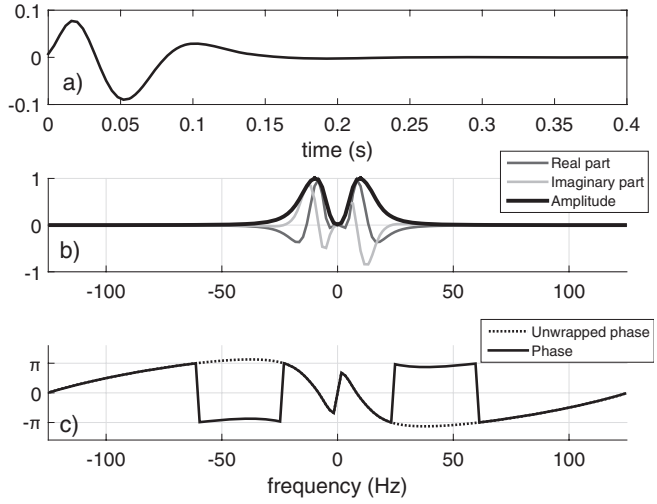


Figure 2.7 A causal wavelet, $f_{\text{dom}} = 10$ Hz, is shown together with its Fourier transform in different representations. (a) The wavelet in the time domain. (b) The real and imaginary parts and the amplitude. The imaginary part is antisymmetric and the other two are symmetric. (c) The phase spectrum, both wrapped and unwrapped. Note the antisymmetry.

the DFT is introduced. For now, just observe the symmetries that have been discussed. The phase spectrum illustrates an additional detail about the computation of phase using Eq. (2.40). The computation of arctan, or $\tan^{-1}$, is done with *atan2*, which always returns a value between $-\pi$ and π. When the actual phase is greater than π by an amount δ, *atan2* returns $-\pi + \delta$ and, similarly, if the actual phase is $-\delta$ less than $-\pi$ it returns $\pi - \delta$. The resulting phase spectrum is said to be *wrapped* and is discontinuous at the points where wrap occurs. MATLAB offers the command *unwrap*, which can remove this effect if the phase spectrum is not too complicated.

Exercises

2.4.7 Repeat the calculations in Exercises 2.4.7 and 2.4.9. In each case identify the real and imaginary parts and the amplitude and phase of the Fourier transforms. Be careful with the Fourier transform of the boxcar. Realize that the amplitude and phase of $+1$ are 1 and 0 but the amplitude and phase of -1 are 1 and π.

2.4.8 Show that the product of two odd functions or of two even functions is even. Similarly, show that an even function times an odd function is odd.

2.4.9 Show that $\int_{-a}^{a} s_{\text{o}}(t)\, dt = 0$, where a is a positive constant and s_{o} is any odd function.

2.4.10 Show that if $s(t)$ is a complex-valued signal then we cannot conclude that the real and imaginary parts of its spectrum have any symmetry.

2.4.11 Show that the spectrum of an even function is purely real and the spectrum of an odd function is purely imaginary.

2.4.3 Some Properties of the Fourier Transform

The inverse Fourier transform gives an exact reconstruction of a signal from its spectrum. Therefore all properties of a signal must have an expression in both domains. Sometimes a given property is easier to express or analyze in one domain than the other. Here we will examine only a few such properties, namely (i) the meaning of 0 Hz, (ii) the dominant frequency, (iii) the time-reversed signal, (iv) calculation of signal energy, (v) the L^1 norm, and (vi) time width versus frequency width.

From the forward Fourier transform, Eq. (2.15), the 0 Hz component of a signal $s(t)$, also called the DC (for direct current) component, is

$$\hat{s}(f = 0) = \int_{-\infty}^{\infty} s(t)\, dt. \tag{2.52}$$

Suppose the signal has a finite length T and is causal; then its mean value is just

$$\bar{s} = \frac{1}{T} \int_0^T s(t)\, dt, \tag{2.53}$$

so we see that for any finite-length signal, $\hat{s}(f = 0)$ is proportional to the mean value of the signal. Most seismic signals oscillate above and below the zero line, so the signal strength at $f = 0$ is very low.

Roughly speaking, the dominant frequency, f_{dom}, of a signal is the "most important" frequency. This can mean different things to different people. The most common choice is to choose f_{dom} as the frequency where the signal's amplitude spectrum reaches its maximum. This works reasonably well if the spectrum is smooth and has a single peak. However, if the spectrum has many peaks, then the simple choice may not be the best. An alternative definition, called the average frequency (or sometimes the centroid frequency) (Cohen, 1995), is given by

$$f_{\text{ave}} = \frac{\int_{-\infty}^{\infty} |f|\, A^p(f)\, df}{\int_{-\infty}^{\infty} A^p(f)\, df} = \frac{\int_0^{\infty} f A^p(f)\, df}{\int_0^{\infty} A^p(f)\, df}, \tag{2.54}$$

where Cohen (1995) chooses $p = 2$, and $A(f) = \left|\hat{s}\right|(f)$ is the signal's amplitude spectrum. The numerator of Eq. (2.54) computes the integral of $|f|$ as weighted by $A^p(f)$, and the denominator normalizes the calculation. The result is an average frequency with respect to $A^p(f)$. The choice of p can affect the results significantly. If the amplitude spectrum has a long tail toward high frequencies, then this can cause f_{ave} to be biased higher than seems reasonable. Larger values of p will downweight the contribution of lower amplitudes, giving more weight to those near local maxima. Figure 2.8 shows measurements of f_{ave} on three different wavelets. The first two wavelets, $w_1(t)$ and $w_2(t)$, have simple spectra with a single peak, while $w_3 = 2 * w_1 + w_2$ is a linear combination of the first two and hence shows two peaks. For each spectrum, there are three estimates, corresponding to $p = 1, 2$, or 3. The measurements can be seen to move toward lower frequencies with increasing p value. Furthermore, the $p = 2$ and $p = 3$ measurements for w_3 can be seen to occur between the two spectral peaks near a local minimum. Nevertheless, they are reasonable

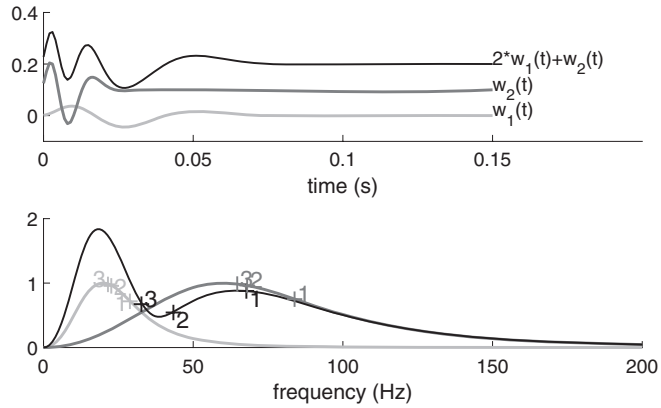


Figure 2.8 Top: Two minimum-phase wavelets, as created by `wavemin`. $w_1(t)$ had $f_{dom} = 20$ Hz and $w_2(t)$ had $f_{dom} = 60$ Hz as prescribed input parameters. $w_3(t)$ is the sum, $w_1 + w_2$. Bottom: The resulting amplitude spectra together with their estimated f_{ave} values for $p = 1, 2$, or 3 in Eq. (2.54). The f_{ave} values are denoted by a + sign and a number, color-coded to identify the spectrum, where the number indicates the p value.

measurements of the average frequency. The measurements were done with the command `domfreq`.

The time-reversed signal plays a recurring role in signal theory. If $s(t)$ is any signal, then its time reverse is simply $s'(t) = s(-t)$. We wish to relate the spectrum of the time-reversed signal to that of the signal. Written as an inverse Fourier transform, the signal is

$$s(t) = \int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df; \tag{2.55}$$

therefore, the time-reversed signal can be expressed as

$$s'(t) = s(-t) = \int_{-\infty}^{\infty} \hat{s}(f) e^{-2\pi i f t}\, df. \tag{2.56}$$

Since $s(-t)$ is real-valued, we can take the complex conjugate of Eq. (2.56) without changing the result, so that

$$s'(t) = \int_{-\infty}^{\infty} \hat{s}^*(f) e^{2\pi i f t}\, df. \tag{2.57}$$

In Eq. (2.56), the minus sign in the exponent means that it is not an inverse Fourier transform, but Eq. (2.57) is. Therefore we can say that the spectrum of the time-reversed signal is the complex conjugate of the signal's spectrum. That is,

$$\hat{s}'(f) = \hat{s}^*(f) = (A(f)e^{i\phi(f)})^* = A(f)e^{-i\phi(f)}, \tag{2.58}$$

where $A(f)$ and $\phi(f)$ are the amplitude and phase of $s(t)$. Thus, time reversal is accomplished in the frequency domain by negation of the phase spectrum. Since the phase

spectrum is antisymmetric, then $-\phi(f) = \phi(-f)$ and so we can also say

$$\hat{s}'(f) = \hat{s}(-f). \tag{2.59}$$

Thus, time reversal and frequency reversal are the same thing.

The energy of a real-valued signal is defined as

$$E = \int_{-\infty}^{\infty} s^2(t)\, dt. \tag{2.60}$$

We wish to find an expression for energy in the frequency domain. This can be done by replacing each $s(t)$ with an inverse Fourier transform,

$$E = \int_{-\infty}^{\infty} \underbrace{\left[\int_{-\infty}^{\infty} \hat{s}(f) e^{2\pi i f t}\, df\right]}_{=s(t)} \underbrace{\left[\int_{-\infty}^{\infty} \hat{s}(f') e^{2\pi i f' t}\, df'\right]}_{=s(t)} dt. \tag{2.61}$$

The reason for using f as the variable of integration in the first square brackets and f' in the second is to emphasize that these expressions do not depend upon the variable of integration. When faced with a complicated multi-integral expression like this, a good practice is to change the order of integration and see what happens. A mathematician would pause to consider if this is a valid step given the assumed properties of the functions being integrated. Indeed, there are cases where interchanging integrations is not correct, but these only arise when integrating very extreme functions. For the kinds of signal relevant to seismology (finite-length and piecewise continuous), we can always exchange the order of integration. In any case, the result which we will obtain has been generalized to more extreme cases. To proceed, we move the t integration to the innermost position:

$$E = \iint_{-\infty}^{\infty} \hat{s}(f)\hat{s}(f') \left[\int_{-\infty}^{\infty} e^{2\pi i (f+f')t}\, dt\right] df\, df'. \tag{2.62}$$

Here the expression in square brackets is equal to $\delta(f+f')$ (see Eq. (2.22)), so that

$$E = \iint_{-\infty}^{\infty} \hat{s}(f)\hat{s}(f')\delta(f+f')\, df\, df' = \int_{-\infty}^{\infty} \hat{s}(f)\hat{s}(-f)\, df, \tag{2.63}$$

where the last step follows from the sifting property of the delta function (Eq. (2.26)). Now, we have just shown that if $\hat{s}(f) = A(f)e^{i\phi(f)}$, then $\hat{s}(-f) = A(f)e^{-i\phi(f)}$. Using this result, we conclude

$$E = \int_{-\infty}^{\infty} \left|\hat{s}(f)\right|^2 df = \int_{-\infty}^{\infty} A^2(f)\, df. \tag{2.64}$$

Comparing Eqs. (2.60) and (2.64) gives the formula known as Parseval's equation,

$$\int_{-\infty}^{\infty} s^2(t)\, dt = \int_{-\infty}^{\infty} \left|\hat{s}(f)\right|^2 df. \tag{2.65}$$

So, the signal energy is the integral of the square of the signal or of the square of the amplitude of the Fourier transform of the signal.

The signal energy computed by use of Parseval's equation is also known as the square of the L^2 norm,[10] denoted by $\|s\|_2$. That is,

$$\|s\|_2 = \left(\int_{-\infty}^{\infty} |s(t)|^2 \, dt\right)^{1/2} = \left(\int_{-\infty}^{\infty} \left|\hat{s}(f)\right|^2 \, df\right)^{1/2} = \left\|\hat{s}\right\|_2, \tag{2.66}$$

where the absolute value has been used in both integrals to allow for the more general case of a complex-valued time-domain signal. The L^2 norm is just one of infinitely many norms of the form $\|s\|_p = \left(\int |s|^p\right)^{1/p}$ and which all measure the "size" of a signal in different ways. While there are infinitely many possible norms, the most important are L^1, L^2, and L^∞. We have seen that $\|s\|_2$ is familiar as the square root of the signal energy and $\|s\|_1$ is an alternative measure that gives less weight to larger signal components (squaring makes larger things larger and smaller ones smaller). The infinity norm $\|s\|_\infty$ gives all weight to the maximum absolute value and is also called the *supremum* or *least upper bound*. Consider the forward Fourier transform of $s(t)$ (Eq. (2.15)) and write the inequality

$$\left|\hat{s}(f)\right| = \left|\int_{-\infty}^{\infty} s(t)e^{-2\pi ift} \, dt\right| \leq \int_{-\infty}^{\infty} \left|s(t)e^{-2\pi ift}\right| \, dt \leq \int_{-\infty}^{\infty} |s(t)| \, dt. \tag{2.67}$$

The first inequality is true because the area beneath an oscillating positive and negative function must be less than the area beneath the absolute value of that function. For the second inequality, we have simply used $\left|s(t)e^{2\pi ift}\right| \leq |s(t)| \left|e^{2\pi ift}\right| = |s(t)|$, which is true since $\left|e^{2\pi ift}\right| = 1$. The right-hand side of the inequality (2.67) is the L^1 norm of $s(t)$, while the left-hand side is the amplitude spectrum of the signal. Since this result must be true for all f, it must hold for the frequency at which $\left|\hat{s}(f)\right|$ assumes its maximum. Thus

$$\max\left|\hat{s}(f)\right| = \left\|\hat{s}\right\|_\infty \leq \|s\|_1 . \tag{2.68}$$

A similar argument allows the conclusion

$$\max|s(t)| = \|s\|_\infty \leq \left\|\hat{s}\right\|_1 . \tag{2.69}$$

So, the L^1 norm in the time domain serves to estimate the maximum absolute amplitude in the frequency domain, and vice versa. While Parseval's theorem says the L^2 norms are equal in both domains, this is not true for other norms. The relations between the L^1 and L^∞ norms will prove useful in the next section.

The final topic of this section is an understanding of the relation between the width of a time-domain signal as compared with the width of its spectrum. Let σ_t represent the width of a time-domain signal and σ_f the width of the corresponding spectrum. Then it turns out that

$$\sigma_t \sigma_f \geq C, \tag{2.70}$$

where C is a constant. In order to make this statement precise, we need to define what we mean by "width" and we also need to estimate C. However, making this statement precise is less important than understanding its general implications for seismic resolution. A

[10] Pronounced "el two norm."

formal proof of the inequality (2.70) can be found in Cohen (1995), where it is called the *uncertainty principle*. This name comes from quantum mechanics, where the waveform analogous to a seismic signal is a "probability" wave that describes the location of a particle. It turns out that the position of a particle and its momentum are always uncertain, with the former being related to the signal width and the latter being related to the width of its Fourier transform. Thus these widths describe uncertainties in fundamental physical quantities (position and momentum), and hence the name of the relation. However, in seismology, our signals are not probability waves but physical waves with measurable properties, and there is no underlying particle; hence the name is not really appropriate for us. A better descriptor would be the "time-width–bandwidth principle," TBP, or the somewhat whimsical "fat–skinny rule."

The TBP, or the inequality (2.70), says that the time width of a signal and the frequency width of its spectrum are inversely proportional. This means that the broader a signal is in time, the narrower it is in frequency, and vice versa. Consider the wavelet w used to construct a convolutional seismogram s in the formula $s(t) = (w \bullet r)(t)$, where r is the reflectivity. When the frequency width of the wavelet is very narrow, the wavelet's time width is large and, when convolved with a reflectivity, the resulting seismogram offers very poor resolution of individual reflection coefficients. Alternatively, if the wavelet's time width is small, then the frequency width is large and the time-domain resolution of individual reflection coefficients is enhanced. (If these statements about resolution are not obvious, then you should review Section 2.3.1.) Figure 2.9 shows four Ormsby wavelets of differing bandwidths in both the time and the frequency domain. The Ormsby wavelet is commonly used in seismology (see Section 1.3) and is specified by four frequencies,

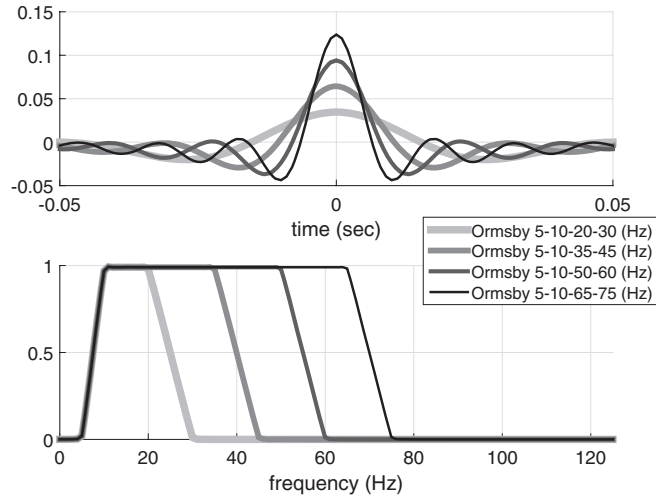


Figure 2.9 Top: Four Ormsby wavelets, each with a different bandwidth, are shown in the time domain. Bottom: The amplitude spectra of the same wavelets. Observe the inverse relationship between time width and frequency width. The wider a wavelet is in time, the narrower it is in frequency, and vice versa. Time width is usually measured as the time between the first zero crossings on either side of the maximum.

usually called f_1, f_2, f_3, and f_4, with the relationship $f_1 < f_2 < f_3 < f_4$. The filter passband (or bandwidth) is the range from f_2 to f_3. A linear ramp from full stop to full pass on the low-frequency end extends from f_1 to f_2, and a similar ramp on the high end extends from f_3 to full stop at f_4. The TBP is evident in Figure 2.9 in that the time widths of the Ormsby wavelets are inversely related to their frequency widths. Measured by the interval between the first zero crossings on either side of the maximum, the time width is greatest for the 5–10–20–30 wavelet, which has the minimum frequency width. Conversely, the maximum frequency width occurs with the 5–10–65–75 wavelet, which has the smallest time width.

Another example of the TBP is found in Exercise 2.4.9, where the reader is asked to show that the Fourier transform of the unit-height boxcar $b_a(t)$ of width $2a$, defined by

$$b_a(t) = \begin{cases} 1, & a \le x \le a \\ 0, & \text{otherwise} \end{cases}, \tag{2.71}$$

is given by

$$\hat{b}_a(f) = 2a\,\mathrm{sinc}(2af) = \frac{\sin(2af)}{f}. \tag{2.72}$$

If not already done, it is advisable to complete Exercise 2.4.9 before proceeding. This involves applying Eq. (2.15) to $b_a(t)$ and realizing that the effect of the boxcar is simply to change the integration limits from $(-\infty, \infty)$ to $[-a, a]$. Then, we decompose $e^{2\pi ift}$ with Euler's identity, discard the antisymmetric part, and complete the integration. If we choose to measure the width of $\hat{b}_a(f)$ by the separation between the first zero crossings on either side of the origin, then this width is $1/a$. Again we see that the time width and bandwidth are inversely proportional to one another.

The Gaussian function plays a special role in the TBP in that the inequality (2.70) becomes an equality when the time-domain function is a Gaussian. That is, a Gaussian of a given time width will have a smaller frequency width than any other function with the same time width. As mentioned previously, Cohen (1995) is an excellent source for technical details on this subject. Here we will simply calculate the Fourier transform of a time-domain Gaussian and examine the result. Let our temporal Gaussian be given by

$$g_\sigma(t) = e^{-t^2/(2\sigma^2)}, \tag{2.73}$$

where σ is the standard deviation of the Gaussian. Cohen (1995) defines the temporal width of a signal $s(t)$ as

$$\sigma_t^2 = \frac{\int_{-\infty}^{\infty}(t-t_0)^2 s^2(t)\,dt}{\int_{-\infty}^{\infty} s^2(t)\,dt}, \tag{2.74}$$

where t_0 is the average time of the signal defined by $t_0 = \int_{-\infty}^{\infty} ts^2(t)\,dt / \int_{-\infty}^{\infty} s^2(t)\,dt$. For the Gaussian of Eq. (2.73), it turns out that $t_0 = 0$ and $\sigma_t = \sigma$. In like fashion, he defines the frequency width as

$$\sigma_f^2 = \frac{\int_{-\infty}^{\infty}(f-f_0)^2 \left|\hat{s}\right|^2 (f)\,df}{\int_{-\infty}^{\infty} \left|\hat{s}\right|^2 (f)\,df}, \tag{2.75}$$

with f_0 defined similarly to t_0. We will not use these definitions in what follows; they are simply here to give a specific definition of width in both domains.

The Fourier transform of $g_\sigma(t)$ is

$$\hat{g}_\sigma(f) = \int_{-\infty}^{\infty} g_\sigma(t) e^{-2\pi i f t}\, dt = \int_{-\infty}^{\infty} e^{-t^2/(2\sigma^2) - 2\pi i f t}\, dt. \tag{2.76}$$

The exponent in the final expression can be simplified by completing the square. That is, $t^2/(2\sigma^2) + 2\pi i f t = (t/(\sqrt{2}\sigma) + \pi i \sqrt{2}\sigma f)^2 + (\pi\sqrt{2}\sigma f)^2$, so that Eq. (2.76) becomes

$$\hat{g}_\sigma(f) = \int_{-\infty}^{\infty} e^{-\left((t/(\sqrt{2}\sigma) + \pi i \sqrt{2}\sigma f)^2 + (\pi\sqrt{2}\sigma f)^2\right) z}\, dt,$$

which becomes

$$\hat{g}_\sigma(f) = e^{-(\pi\sqrt{2}\sigma f)^2} \int_{-\infty}^{\infty} e^{-(t/(\sqrt{2}\sigma) + \pi i \sqrt{2}\sigma f)^2}\, dt. \tag{2.77}$$

Now, let $u = t/(\sqrt{2}\sigma) + \pi i \sqrt{2}\sigma f$ and $du = dt/(\sqrt{2}\sigma)$, so

$$\hat{g}_\sigma(f) = \sqrt{2}\sigma e^{-(\pi\sqrt{2}\sigma f)^2} \int_{-\infty}^{\infty} e^{-u^2}\, du. \tag{2.78}$$

This final integral is found in most tables of definite integrals and equals $\sqrt{\pi}$. Thus

$$\hat{g}_\sigma(f) = \sqrt{2\pi}\sigma e^{-(\pi\sqrt{2}\sigma f)^2}. \tag{2.79}$$

So, we have the very interesting result that a Gaussian transforms into a Gaussian. The original time-domain Gaussian had a width (standard deviation) of $\sigma_t = \sigma$. Inspection of Eq. (2.79) shows that it has a standard deviation of $\sigma_f = (2\pi\sigma)^{-1}$. Thus the product of the widths is $\sigma_t\sigma_f = 1/(2\pi)$. Again, the important thing here is not the precise value of $\sigma_t\sigma_f$ but rather that this means that σ_t and σ_f are inversely proportional, which is another manifestation of the TBP. Cohen (1995) shows that the Gaussian has the smallest possible value of $\sigma_t\sigma_f$, which means that the Gaussian is often an optimum window (multiplier) in either domain.

Exercises

2.4.12 Consider Figures 2.10a and 2.10b and notice that the various boxcars all have the same height, while the corresponding sinc functions have a low maximum amplitude when the boxcar is narrow and this increases as the boxcar becomes wider. The same effect can be observed in Figure 2.9, where the Ormsby amplitude spectra all have the same height but the corresponding time-domain wavelets progress from small to large. Explain why this must be so. What property of the Fourier transform requires this to be so?

2.4.4 The Convolution Theorem and the Differentiation Theorem

The Fourier transform has great importance in seismology primarily because it allows us to reexpress convolution and differentiation as simpler operations in the Fourier domain.

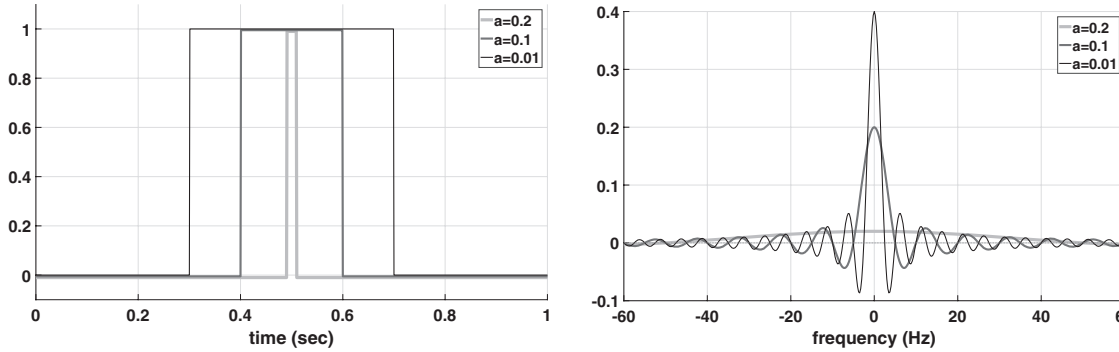


Figure 2.10a (left) Three unit-height boxcars given by Eq. (2.71) are shown for different values of *a*. Slight vertical shifts have been applied in plotting so that the functions do not overlap.

Figure 2.10b (right) The Fourier transforms of the three boxcars of Figure 2.10a, as described by Eq. (2.72). Comparison of these two figures shows the TBP: the wider the boxcar, the narrower the sinc, and vice versa.

For convolution, this leads to fast methods of filtering and wavelet application, while for differentiation, new methods of solving the wave equation become available.

Consider first the convolution $s(t) = (u \bullet v)(t)$, which has the integral form

$$s(t) = \int_{-\infty}^{\infty} u(t-\tau)v(\tau)\,d\tau. \tag{2.80}$$

We will show that the spectrum of $s(t)$ is the product of the spectra of $u(t)$ and $v(t)$. To prove this, we substitute into Eq. (2.80) the definitions of $u(t-\tau)$ and $v(\tau)$ as inverse Fourier transforms of their spectra. This gives

$$s(t) = \int_{-\infty}^{\infty} \underbrace{\left[\int_{-\infty}^{\infty} \hat{u}(f)e^{2\pi if(t-\tau)}\,df\right]}_{u(t-\tau)} \underbrace{\left[\int_{-\infty}^{\infty} \hat{v}(f')e^{2\pi if'\tau}\,df'\right]}_{v(\tau)} d\tau, \tag{2.81}$$

where, as in the previous section, we are careful to use different symbols f and f' as frequencies in the two inverse transforms. Now, we move the τ integral to the inside:

$$s(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \hat{u}(f)\hat{v}(f')e^{2\pi ift}\left[\int_{-\infty}^{\infty} e^{2\pi i(f'-f)\tau}\,d\tau\right] df\,df'. \tag{2.82}$$

The quantity in square brackets is $\delta(f'-f)$ and the delta function sifting rule (Eq. (2.26)) then collapses the f' integral to give

$$s(t) = \int_{-\infty}^{\infty} \hat{u}(f)\hat{v}(f)e^{2\pi ift}\,df. \tag{2.83}$$

Recognizing this integral as an inverse Fourier transform proves the basic result that $\hat{s}(f) = \hat{u}(f)\hat{v}(f)$, and this is called the *convolution theorem*.

The mathematics shown in Eqs. (2.81)–(2.82) requires considerable effort to justify with full mathematical rigor. There are subtle but difficult questions such as "For what class of functions do the various integrals converge?" and "Under what circumstances may the order of integration be changed?" These questions have been answered, and the operations are valid for a very general class of functions and even distributions. The interested reader should consult a treatise on *harmonic analysis* such as Korner (1988) or Stein (1993). For this book, it will be assumed that the convolution theorem is valid for all functions encountered.

The importance of this result is that it means that a filter can be applied to a signal in the Fourier domain by multiplying the spectrum of the signal by that of the filter. For digital computations, this is often faster than direct evaluation of the convolution integral because of the *fast Fourier transform* algorithm. However, there is more to this story that will be discussed after the *discrete Fourier transform* is introduced.

The symmetry of the Fourier transform means that the time and frequency domains are essentially similar. Thus it follows that the convolution theorem also works in reverse. That is, the product $u(t)v(t)$ is the convolution of the spectra of these functions in the frequency domain, with the result

$$\int_{-\infty}^{\infty} u(t)v(t)e^{-2\pi ift}\,dt = \int_{-\infty}^{\infty} \hat{u}(f')\hat{v}(f-f')\,df'. \tag{2.84}$$

An immediate consequence of these results is that a temporally short signal must have a smooth spectrum. Consider a signal $s(t)$ that is nonzero only for t in the interval $0 \le t \le T$. Let $w_T(t)$ be any function that is unity in the same time interval and which tapers smoothly but rapidly to 0 outside this interval. Then we have the identity $s(t) = w_T(t)\,s(t)$. In this context, $w_T(t)$ is an example of a temporal windowing function which is applied to the signal by multiplication. It must therefore be true that $\hat{s}(f) = \left(\hat{s} \bullet \hat{w}_T\right)(f)$. Since we know that convolution has a smoothing effect, then there must exist a family of convolutional smoothers, one for each possible w_T, which do not change $\hat{s}(f)$. This can only be possible if $\hat{s}(f)$ is already smooth. Moreover, the smaller T is, the smoother the spectrum of $s(t)$. The argument can also be constructed in reverse to show that a band-limited function cannot have discontinuities. We say that $s(t)$ is a band-limited function if there exists a positive constant F such that $\hat{s}(f) = 0$ for $|f| > F$. Then, let $w_F(f)$ be a function that is unity on the interval $-F \le f \le F$ and which tapers to zero smoothly and rapidly outside this interval. Then we must have $\hat{s}(f) = \hat{s}(f)w_F(f)$, which means that $s(t) = \left(s \bullet \check{w}_F\right)(t)$, where $\check{w}_F$ is the inverse Fourier transform of w_F. So again there must be a class of convolutional smoothers which leave $s(t)$ unchanged, and thus $s(t)$ must already possess a degree of smoothness, which implies that it cannot have discontinuities. When we study the wave equation, we will see that all seismic signals must be band limited, so that there cannot be discontinuities in a real seismic wavefield. These arguments are more intuitive than mathematically rigorous. However, the differentiation theorem will allow a more precise argument.

Also of great importance is that the Fourier transform reduces differentiation to multiplication. For example, consider the calculation of $ds(t)/dt$ given $s(t)$ as an inverse Fourier

transform of its spectrum,

$$\frac{ds(t)}{dt} = \frac{d}{dt}\int_{-\infty}^{\infty} \hat{s}(f)e^{2\pi ift}\,df. \tag{2.85}$$

Examination of the right-hand side of this equation shows that the t dependence occurs only in the exponent of the complex exponential. Therefore the differentiation can be moved under the integration sign and performed analytically, with the result

$$\frac{ds(t)}{dt} = \int_{-\infty}^{\infty} 2\pi if\hat{s}(f)e^{2\pi ift}\,df. \tag{2.86}$$

Thus differentiation with respect to t can be performed by multiplying the spectrum of $s(t)$ by $2\pi if$ or $i\omega$. In this case, the angular frequency leads to a simpler formula. This result is called the *differentiation theorem*. When the differentiation is in the frequency domain, a similar argument gives

$$\frac{d\hat{s}(f)}{df} = \int_{-\infty}^{\infty} (-2\pi it)s(t)e^{-2\pi iift}\,dt. \tag{2.87}$$

More general differentiation operators can also be computed. Let $a(t, d/dt) = b_1(t)d/dt + b_2(t)(d^2/dt^2)$ be a differential operator; then

$$\begin{aligned} a\left(t, \frac{d}{dt}\right)s(t) &= b_1(t)\frac{ds(t)}{dt} + b_2(t)\frac{d^2s(t)}{dt^2} = \frac{1}{2\pi}\int_{-\infty}^{\infty}\left[i\omega b_1(t) - \omega^2 b_2(t)\right]\hat{s}(\omega)e^{i\omega t}\,d\omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\alpha(t,\omega)\hat{s}(\omega)e^{i\omega t}\,d\omega, \end{aligned} \tag{2.88}$$

where $\alpha(t, \omega) = i\omega b_1(t) - \omega^2 b_2(t)$ is called the *symbol* of the differential operator. In this manner, virtually any differential operator can be converted into an equivalent Fourier-domain multiplicative operator. This observation means that the Fourier transform is of great importance in solving partial differential equations.

Consider again the relation between the temporal length of a signal and the smoothness of its spectrum. Using the derivative, a measure of spectral smoothness is $\left\|d\hat{s}(f)/df\right\|_\infty$, meaning that a smooth function will have a small $\max\left|d\hat{s}(f)/df\right|$. Using Eq. (2.87) and the inequality (2.68), we have

$$\left\|\frac{d\hat{s}(f)}{df}\right\|_\infty \le 2\pi\int_{-\infty}^{\infty}|ts(t)|\,dt = 2\pi\int_0^T |ts(t)|\,dt, \tag{2.89}$$

where we have assumed $s(t)$ to be nonzero only for $0 \le t \le T$. Using $|t| \le T$, we can rewrite this result as

$$\left\|\frac{d\hat{s}(f)}{df}\right\|_\infty \le 2\pi T\int_0^T |s(t)|\,dt = 2\pi T\left\|s\right\|_1. \tag{2.90}$$

So, the smaller T is, the smaller $\left\|d\hat{s}(f)/df\right\|_\infty$ is and the smoother is the spectrum. In fact, a similar argument shows that the nth derivative is bounded by $(2\pi T)^n\left\|s\right\|_1$. This argument

also works in reverse so, as previously mentioned, a band-limited function is smoothly continuous in the time domain.

Exercises

2.4.13 Derive Eq. (2.84).

2.4.14 Let $s(t) = (u \bullet v)(t)$. Derive an expression for the amplitude and phase spectra of $s(t)$ in terms of the amplitude and phase spectra of $u(t)$ and $v(t)$.

2.4.15 Equation (2.86) shows that differentiation can be accomplished by multiplication in the Fourier domain. Therefore, it must be possible to accomplish differentiation by a generalized convolutional operator. Or, alternatively, it makes sense to talk about a differentiation filter. What are the amplitude and phase of the differentiation filter for ∂_t? (Give results for both positive and negative frequencies.) Is this filter band limited?

2.4.16 Let $s_I(t) = \int_0^t s(t')\,dt'$ define the indefinite integral (antiderivative) of the signal $s(t)$. Show that integration is accomplished in the Fourier domain by the multiplier $-i/(2\pi f)$. What are the amplitude and phase of the integration filter?

2.4.17 Let $s_n(t) = s(t) + n(t)$ be a real-world signal that contains noise, where $s(t)$ is a noise-free signal and $n(t)$ is "white" noise. White noise has the property that $\left|\hat{n}(f)\right| \approx C$, where C is a constant. Suppose further that the maximum of $\left|\hat{s}(f)\right|$ is much greater than C but that $\left|\hat{s}(f)\right|$ decays with increasing $|f|$ so that there will always exist some frequency F such that $\left|\hat{s}(f)\right| < C, |f| > F$. Consider the derivative $\partial_t s_n(t)$. Will the derivative of s_n be more or less noisy than $s_n(t)$? Why? Repeat these considerations for the indefinite integral of $s(t)$.

2.4.5 The Phase Shift Theorem

Another important result involving Fourier transforms is the relationship between a time shift and a phase shift. Consider the Fourier transform of the time-shifted signal $s_{\Delta t}(t) = s(t + \Delta t)$

$$\hat{s}_{\Delta t}(f) = \int_{-\infty}^{\infty} s(t + \Delta t) e^{-2\pi i f t}\,dt \tag{2.91}$$

and make the change of variables $\tau = t + \Delta t$ so that

$$\hat{s}_{\Delta t}(f) = \int_{-\infty}^{\infty} s(\tau) e^{-2\pi i f(\tau - \Delta t)}\,d\tau = e^{2\pi i f \Delta t} \int_{-\infty}^{\infty} s(\tau) e^{-2\pi i f \tau}\,d\tau = e^{2\pi i f \Delta t} \hat{s}(f). \tag{2.92}$$

In the final expression, $\hat{s}(f)$ is the Fourier transform of $s(t)$. This shows that the spectrum of the time-shifted signal may be computed from the spectrum of the original signal by multiplication by $e^{2\pi i f \Delta t}$. This operator has unit amplitude and a phase that is linear in f with a slope $2\pi\ \Delta t$. Let $\hat{s}(f) = A_s(f) e^{i\phi_s(f)}$, and Eq. (2.92) becomes

$$\hat{s}_{\Delta t}(f) = \hat{s}(f) e^{2\pi i f \Delta t} = A_s(f) e^{i\phi_s(f) + i2\pi f \Delta t}. \tag{2.93}$$

Thus, the spectrum of the time-shifted signal is obtained from the spectrum of the original signal by a linear *phase shift*. That is, a linear (in f) term is added to the phase. The magnitude of the slope of the linear term is directly proportional to the magnitude of the time shift.

One subtlety is that the overall sign of the phase shift depends upon the sign convention used in the definition of the Fourier transform. That is, Eq. (2.93) results from the Fourier transform pair of equations (2.15)–(2.16). If the sign convention is reversed so that the forward transform is done with $\exp(2\pi ift)$, then the sign of the phase shift required for a positive time shift will be reversed. Thus, if a theory is derived analytically with one sign convention and then implemented with software that uses the other sign convention, confusing results can ensue. As a practical rule, if a phase shift program does not give the expected results, then it should be tested with the sign of the phase reversed.

The phase shift theorem is especially important with sampled data. This is because a time shift which is not an integral number of samples requires that new sample values be interpolated from the signal. This is because a sampled signal must always have values for the same set of time samples $0, \Delta t, 2\,\Delta t, 3\,\Delta t, \ldots$, where Δt is the time sample interval. If the samples before the time shift are $s_0, s_1, s_2, \ldots$, then a time shift of exactly Δt will give samples like $0, s_0, s_1, s_2, \ldots$. That is, the first sample becomes the second, the second becomes the third, and so on. However, suppose a shift of $0.5\,\Delta t$ is desired. Then the sample needed at time Δt is halfway between s_0 and s_1; in fact, none of the samples of the shifted trace are present in the input samples. This difficultly can be met by interpolating sample values at locations halfway between the input samples. However, proper interpolation of sampled signals is tricky (see Section 3.2). If the time shift is done as a phase shift, then this interpolation is done implicitly and very accurately through the machinery of the Fourier transform.

The phase shift theorem can also help us to understand why the phase spectrum of most signals is a very complicated thing. In Section 2.4.2, the phenomenon of phase wrapping was discussed. This refers to the fact that our method of phase computation using *atan2* must always return a value between $-\pi$ and π regardless of how large the phase actually is. Fundamentally, this issue arises because $\tan\theta$, like all of the trigonometric functions, has periodicity. As shown in Figure 2.11a, the tangent has a central period from $-\pi/2$ to $\pi/2$ and repeats itself every π radians. Recall that the tangent is usually computed as a ratio like $\tan\theta = a/b$. If only the ratio is known, then the only possibility for $\tan^{-1}$ is the function *atan*, which takes a single input a/b and returns an angle in the range $-\pi/2$ to $\pi/2$. However, if both a and b are known, as is the case for the phase computation, then *atan2* can be used, which returns an angle in the range $-\pi$ to π. Thus, with *atan2* we get twice the range and can resolve angles all the way around the unit circle. However, the phase shift required for a given time shift is $\phi(f) = 2\pi f\,\Delta t$, which quickly exceeds π for very modest values of f and Δt. As Figure 2.11b shows, a minimum-phase wavelet with an initially continuous phase rapidly achieves a phase far greater than π with even very small time shifts. The message here is that phase can be regarded as position and, for example, a position of $t = 1$ s with a frequency of 10 Hz requires a phase of $\phi(f) = 20\pi$, which is far in excess of the range of *atan2*. When calculated with the inverse Fourier transform, such a phase is always projected into the range of *atan2* and this results in a very discontinuous

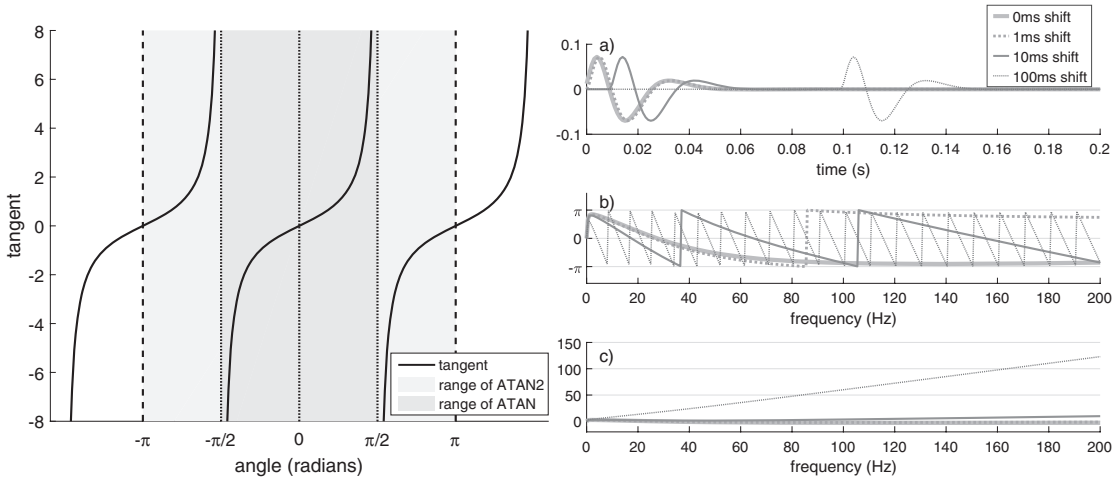


Figure 2.11a (left) The tangent function is shown over the angle domain from $-3\pi/2$ to $3\pi/2$. The function *atan* takes a single input argument and returns an angle in the range from $-\pi/2$ to $\pi/2$. A better choice is *atan2*, which takes two inputs and returns an angle in the range $-\pi$ to π.

Figure 2.11b (right) (a) A minimum-phase wavelet is shown together with three time-shifted copies of itself. The time shifts were accomplished by phase shift. (b) The phases of the four wavelets in panel a as computed by *atan2*. (c) The actual phases $\phi(f) = 2\pi f\,\Delta t$ used to time-shift the wavelets in panel a.

function, which can look chaotic. Imagine a reflectivity function formed by placing each reflection coefficient at the origin, where its phase will be either 0 or π, phase-shifting it to its proper location in time, and then superimposing the results for all reflection coefficients. The result will be a function whose wrapped phase will be almost impossible to understand. Thus, quite generally, amplitude spectra are relatively easy to interpret and understand, while phase spectra can be baffling.

2.4.6 Phase Rotations

A common task in seismic processing is the phase rotation of a signal. This can be accomplished by a multiplication in the Fourier domain by a suitable *phase rotation function*. Such a multiplier is defined by the requirements that it must add a constant (independent of frequency) to the phase at each frequency and that the rotated signal must be real if the original signal is real. For a phase rotation by an angle of θ radians, the rotation is accomplished in the Fourier domain by

$$\hat{s}_\theta(f) = \hat{s}(f)e^{i\,\mathrm{sgn}(f)\theta}, \tag{2.94}$$

where $\mathrm{sgn}(f) = f/|f|$ (with $\mathrm{sgn}(0) = 1$) is the *sign* function for frequency and $\hat{s}(f)$ is the spectrum of the unrotated signal. Considering the polar decomposition of the spectrum (Eq. (2.38)), it is apparent that Eq. (2.94) adds the constant θ to the phase of the positive frequencies and $-\theta$ to the phase of the negative frequencies. This handling of the negative

and positive frequencies differently is required to maintain the Hermitian symmetry of the spectrum of a real-valued signal (Eq. (2.51)). Using Euler's identity, the phase rotation expression can be rewritten as

$$\hat{s}_\theta(f) = \hat{s}(f)\left[\cos(\mathrm{sgn}(f)\theta) + i\sin(\mathrm{sgn}(f)\theta)\right] = \hat{s}(f)\left[\cos(\theta) + i\,\mathrm{sgn}(f)\sin(\theta)\right], \tag{2.95}$$

where the last form follows from the fact that the cosine is an even function and the sine is an odd function. Taking the inverse Fourier transform of Eq. (2.95) leads to

$$s_\theta(t) = \cos(\theta)s(t) + \sin(\theta)s_\perp(t) = \cos(\theta)s(t) - \sin(\theta)\mathcal{H}[s(t)], \tag{2.96}$$

where

$$s_\perp(t) = \int_{-\infty}^{\infty} i\,\mathrm{sgn}(f)\hat{s}(f)e^{2\pi ift}\,df = \int_{-\infty}^{\infty} e^{i\,\mathrm{sgn}(f)\pi/2}\hat{s}(f)e^{2\pi ift}\,df \tag{2.97}$$

is the 90° phase-rotated trace and $\mathcal{H}[s(t)] = -s_\perp(t)$ is the *Hilbert-transformed* trace, defined by

$$\mathcal{H}[s(t)] = \int_{-\infty}^{\infty} (-i\,\mathrm{sgn}(f))\hat{s}(f)e^{2\pi ift}\,df. \tag{2.98}$$

In the definition in Eq. (2.98), the Hilbert transform of a time-domain signal is accomplished in the Fourier domain through multiplication by $-i\,\mathrm{sgn}(f)$. Since $-i\,\mathrm{sgn}(f) = e^{-i\,\mathrm{sgn}(f)\pi/2}$ is a Fourier rotation multiplier for $-90°$, the Hilbert-transformed signal $\mathcal{H}[s(t)]$ is a $-90°$ rotated signal. Thus Eq. (2.96) has the interesting interpretation that an arbitrary phase rotation of the signal $s(t)$ is represented as a linear combination of $s(t)$ and its 90° rotation, $s_\perp(t)$. The Hilbert transform can also be written in the time domain as a convolution of the signal with the inverse Fourier transform of $-i\,\mathrm{sgn}(f)$. The inverse Fourier transform of $\mathrm{sgn}(f)$ is well known to be $i/(\pi t)$ (e.g., Karl (1989), p. 119, or Bracewell (2000), p. 139), so the frequency-domain Hilbert transform can also be written as a time-domain convolution of the signal with $1/(\pi t)$,

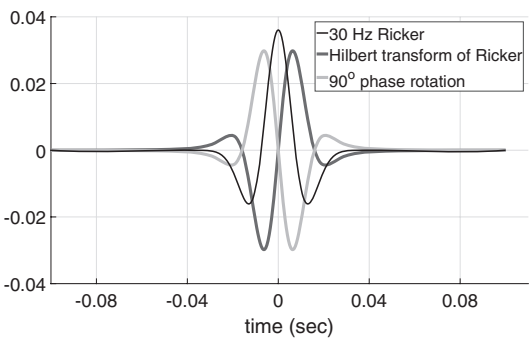
$$\mathcal{H}[s(t)] = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{s(t')}{t - t'}\,dt'. \tag{2.99}$$

The evaluation of Eq. (2.99) may require the use of the Cauchy principal value to handle any potential singularity at the origin $t = t'$. In MATLAB the Hilbert transform is accomplished with the `hilbert` command, as in `sa=hilbert(s)`, which produces a new signal, s_a, called the analytic signal, whose real part is the original signal, s, and whose imaginary part is $\mathcal{H}[s(t)]$. More will be said about the analytic signal shortly.

As an example of phase rotations, let us calculate a series of phase rotations for the Ricker wavelet. This wavelet is a standard model for a zero-phase seismic wavelet with a particular dominant frequency. The time-domain expression for the Ricker wavelet is

$$w(t) = \left(1 - 2\pi^2 f_{\mathrm{dom}}^2 t^2\right) e^{-(\pi^2 f_{\mathrm{dom}}^2 t^2)}, \tag{2.100}$$

where f_{dom} is the dominant frequency in Hz. To within a constant scale factor, the Ricker wavelet is the second derivative of the Gaussian $e^{-(\pi^2 f_{\mathrm{dom}}^2 t^2)}$. Figure 2.12a shows $w(t)$,



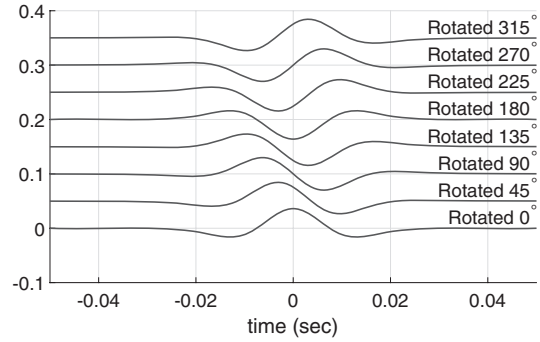


Figure 2.12a (left) A 30 Hz Ricker wavelet is shown with its Hilbert transform, which is a $-90°$ phase rotation. Note that the $\pm 90°$ rotations put zero crossings where there were formerly maxima or minima, and vice versa.

Figure 2.12b (right) A series of phase rotations that were calculated from Eq. (2.96) using the wavelets in Figure 2.12a. Each successive phase rotation is shifted upward by 0.5 to avoid overplotting.

$\mathcal{H}[w(t)]$, and $w_\perp(t)$. These three closely related wavelets, while similar in overall magnitude, have distinctly different appearances. At the times of the maxima and minima of $w(t)$, $w_\perp(t)$ and $\mathcal{H}[w(t)]$ have zero crossings, while the extrema of the $\pm 90°$ phase rotations correspond to the zero crossings of w. Also, w is symmetric about $t = 0$, while both $\pm 90°$ phase rotations are antisymmetric. These properties are similar to those found in a comparison between the cosine and sine functions, and in fact the latter is the Hilbert transform of the former. Some results obtained when Eq. (2.96) is used to generate other phase rotations are shown in Figure 2.12b. Comparison of these shows that a phase rotation of $\pm 90°$ causes the aforementioned correspondence between extrema and zero crossings, while a phase rotation of $\pm 180°$ causes a flip in polarity.

A major use of the Hilbert transform is in the computation of the *trace envelope*, as mentioned in Section 1.4.1. We first construct the *analytic signal*, defined by

$$s_a(t) = s(t) + i\mathcal{H}[s(t)] = s(t) - is_\perp(t). \tag{2.101}$$

The analytic signal of a real signal is complex-valued, with the imaginary part formed from the Hilbert transform of the real part. An interesting property of the analytic signal is that its Fourier transform vanishes for negative frequencies. This is verifiable by direct calculation, for it follows from the linearity of the Fourier transform, and the fact that the Hilbert transform in the Fourier domain is multiplication by $-i\,\mathrm{sgn}(f)$, that the Fourier transform of the analytic signal, called $\hat{s}_a(f)$, is

$$\hat{s}_a(f) = \hat{s}(f) + \mathrm{sgn}(f)\hat{s}(f) = [1 + \mathrm{sgn}(f)]\,\hat{s}(f). \tag{2.102}$$

Clearly, $1 + \mathrm{sgn}(f)$ vanishes for $f < 0$ and equals 2 for $f > 0$. Thus the analytic trace has a spectrum that is identically zero for negative frequencies and is double the spectrum of the original trace for positive frequencies.

From the analytic trace defined in Eq. (2.101), we define the *trace envelope* (also called the *instantaneous amplitude*) as

$$\epsilon_s(t) = |s_a(t)| = \sqrt{s(t)^2 + \mathcal{H}[s(t)]^2} = \sqrt{s(t)^2 + s_\perp(t)^2}. \tag{2.103}$$

The trace envelope has the important property

$$\epsilon_s(t) \geq |s_\theta(t)| \tag{2.104}$$

for any θ, where $s_\theta(t)$ is a constant-phase rotation of $s(t)$ as specified by Eq. (2.96). To prove the inequality (2.104), define two abstract vectors $U = (s, s_\perp)$ and $V = (\cos\theta, \sin\theta)$ and note that $s_\theta = s\cos\theta + s_\perp \sin\theta = U \cdot V$, where $U \cdot V$ is the vector dot product, or inner product, of U and V. The Schwarz inequality, which is true for any two vectors in a space with an inner product defined on it, says that

$$(U \cdot V)^2 \leq (U \cdot U)(V \cdot V). \tag{2.105}$$

For our definitions of U and V we have

$$(U \cdot V)^2 = (s\cos\theta + s_\perp \sin\theta)^2 \leq \underbrace{\left(s^2 + s_\perp^2\right)}_{U \cdot U} \underbrace{\left(\cos^2\theta + \sin^2\theta\right)}_{V \cdot V} = s^2 + s_\perp^2 = \epsilon_s^2, \tag{2.106}$$

and taking the square root gives the inequality (2.104). As a numerical demonstration of Eq. (2.104), consider the code in Code Snippet 2.4.1, which produces Figures 2.13a and 2.13b.

The enclosing property of the envelope is true for all constant-phase rotations of any signal, not just the Ricker wavelet. Figure 2.14 demonstrates this for a simple synthetic seismogram using a code similar to Code Snippet 2.4.1. The seismogram was created by convolving a 30 Hz Ricker wavelet with a reflectivity created with the `reflec` command. While all constant-phase rotations are contained within the envelope, it does not follow that a nonconstant phase shift will result in a signal that lies within the envelope. As a simple example, consider a linear (with frequency) phase shift. As discussed in Section 2.4.5, such a phase shift corresponds to a constant time shift, which can clearly move energy outside the envelope. Nevertheless, the trace envelope is often used as a measure of trace energy that is less sensitive to the phase of the wavelet than the trace itself. Seismic interpreters will sometimes pick horizon locations at the nearest peak of the Hilbert envelope, assuming that such a pick will be robust if the wavelet has an unknown phase error.

The trace envelope is the simplest of a suite of quantities, derivable from the analytic trace, that are collectively known as the *Hilbert attributes*. The other common attributes, which are derivable from the instantaneous amplitude, are the *instantaneous phase* and *instantaneous frequency*. These quantities have found use mostly in seismic interpretation and will not be discussed further here. The interested reader should consult Cohen (1995) or Barnes (2007) for a detailed discussion.

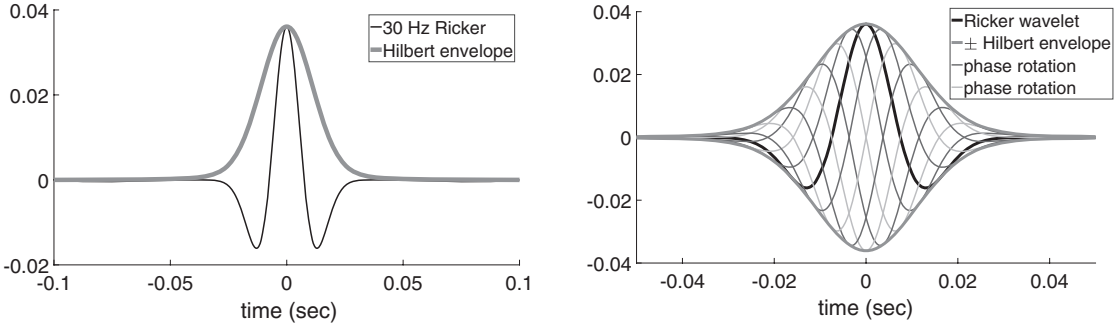


Figure 2.13a (left) A 30 Hz Ricker wavelet is shown with its Hilbert envelope. The Hilbert envelope is the absolute value (or magnitude) of the analytic trace and is given by Eq. (2.103).

Figure 2.13b (right) For the wavelet at the left, phase rotations for angles of 0, 45, and 315° are shown together with the Hilbert envelope. The envelope is shown as both $+|sa|$ and $-|sa|$, where sa is the analytic signal derived from the wavelet. The original wavelet is the bold black line. Note that the axis limits are different from those of Figure 2.13a.

Code Snippet 2.4.1 This example creates a Ricker wavelet (the `ricker` command) and then calculates the analytic trace (called `wa`), the Hilbert transform (`wh`), and a series of phase rotations. These are then plotted to demonstrate Eq. (2.104). The result of this code is Figure 2.13b.

```
dt=.0005;%time sample rate
tlen=.2;%wavelet length
fdom=30;%dominant frequency
[w,tw]=ricker(dt,fdom,tlen);
wa=hilbert(w);%analytic signal corresponding to Ricker
wh=-imag(wa);%Hilbert transform of Ricker
env=abs(wa);
angles=[0:45:315];%phase rotation angles
wrot=zeros(length(w),length(angles));%preallocate space
for k=1:length(angles)
    wrot(:,k)=w*cosd(angles(k))+wh*sind(angles(k));%phase rotations
end
figure
hh=plot(tw,wrot,tw,env,'r',tw,-env,'r');
xlabel('time (sec)')
xlim([-.1 .1])
prepfig
set(hh(1),'linewidth',5,'color','k')
```

End Code

signalcode / rickerenvrot .m

Exercises

2.4.18 Create a convolutional synthetic seismogram using a Ricker wavelet having a dominant frequency of 30 Hz and a reflectivity created by `reflec`, and calculate

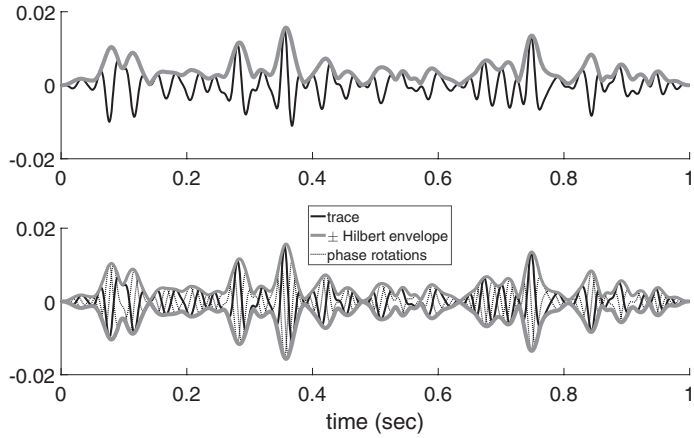


Figure 2.14 Top: A synthetic seismogram (trace) formed by convolving a 30 Hz Ricker wavelet with a synthetic random reflectivity is shown with its Hilbert envelope. Bottom: Similarly to Figure 2.13b, the trace of panel a is shown together with its positive and negative envelope and the phase rotations 90° and 270°. All of the phase rotations are contained within the envelope as Eq. (2.104) requires.

the corresponding analytic trace, the Hilbert transform trace, and the trace envelope. Then create a series of constant-phase rotations and make a figure similar to Figure 2.14. Finally, create a minimum-phase wavelet with the same dominant frequency and use this to generate another synthetic seismogram from the same reflectivity. Is this new seismogram contained within the envelope generated from the first seismogram? Is it fair to say that the envelope is independent of phase? Discuss.

2.4.19 Write a MATLAB function that takes two inputs, a signal and an angle in degrees, and returns as output the input signal rotated in phase by the input angle. You may use a Fourier approach or a Hilbert transform method. Test your code with a number of different angles. Does a 180° phase shift give a polarity flip? Compare a 90° rotation with the original signal. What feature on the original signal is at the same time as a peak or trough on the 90° rotated signal?

2.4.7 The Spectrum of a Causal Signal

In Section 2.4.2, it was demonstrated that a real-valued signal imposes a certain symmetry upon its spectrum. It is a similar story with a causal signal, that is, a function $s(t)$ for which $s(t) = 0$ when $t < 0$. In the causal case, the real and imaginary parts of the Fourier spectrum $\hat{s}(\omega)$ are related via the Hilbert transform. This symmetry is described more fully in the following.

The causal symmetry can be derived from the observation that a signal $s(t)$ is causal if and only if its Fourier transform $\hat{s}(\omega)$ extends to an analytic function in the lower half of

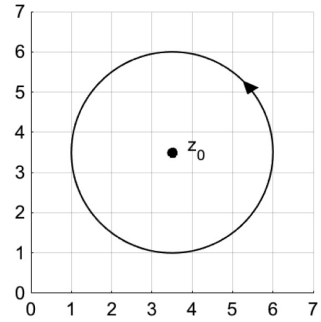


Figure 2.15 A simple closed curve for the Cauchy integral formula.

the complex plane by the formula

$$\hat{s}(z) = \int_0^\infty s(t)e^{-izt}\,dt, \tag{2.107}$$

where $z = x + iy$ is the complex variable, with $y < 0$. This integral defines an analytic function since, on the lower half-plane, the analytic exponentials $e^{-izt} = e^{-ixt}e^{yt}$ decay exponentially fast as $y \mapsto -\infty$, so the integral converges rapidly, preserving the analytic structure. The usual Fourier transform of $s(t)$ is recovered for real values of z via the formula $\hat{s}(x) = \hat{s}(x + i0)$, since there is no contribution to the transform for negative time.

Analytic functions such as $\hat{s}(z)$ have the remarkable property that their value at any point z_0 can be recovered by a contour integral along a simple closed loop Γ containing that point, via the Cauchy integral formula

$$\hat{s}(z_0) = \frac{1}{2\pi i}\oint_\Gamma \frac{\hat{s}(z)}{z - z_0}\,dz. \tag{2.108}$$

In the special case where the curve Γ is a circle of radius R, as in Figure 2.15, Cauchy's formula follows from a power series expansion of the function $\hat{s}(z) = a_0 + a_1(z - z_0) + a_2(z - z_0)^2 + \cdots$. Parameterizing the circle about z_0 as $z = z_0 + Re^{it}$, $dz = iRe^{it}dt$, for $t \in [0, 2\pi]$, we see that almost all the terms in the power series lead to complex exponentials, which are sines and cosines that integrate to zero. The only exception is the a_0 term, which gives us

$$\frac{1}{2\pi i}\oint_\Gamma \frac{\hat{s}(z)}{z - z_0}\,dz = \frac{1}{2\pi i}\oint_\Gamma \frac{a_0}{z - z_0}\,dz = \frac{1}{2\pi i}\int_0^{2\pi} \frac{a_0}{Re^{it}} iRe^{it}\,dt = \frac{1}{2\pi}\int_0^{2\pi} a_0\,dt = a_0, \tag{2.109}$$

which is the desired result since, by the series expansion again, $\hat{s}(z_0) = a_0$.

The more general case of the Cauchy integral formula for a simple closed curve Γ follows from Green's theorem, where we can continuously deform the curve Γ into a simple

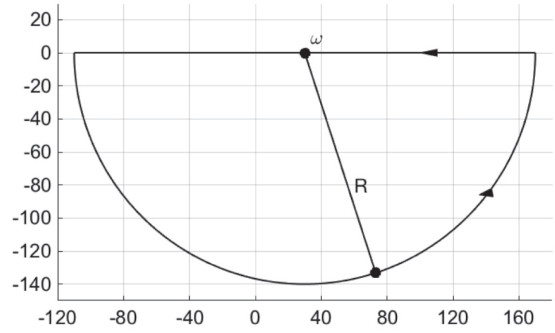


Figure 2.16 A half-circle path on the lower half-plane.

circle, and not change the integral in Eq. (2.108) provided we do not pass through any singularities.

We use Eq. (2.108) to compute the value $\hat{s}(\omega)$ at a point ω on the x axis, choosing the closed curve Γ in Figure 2.16 that runs along the x axis from ∞ to $-\infty$ and loops counterclockwise around the lower half at some radius R. The resulting integral formula gives

$$\hat{s}(\omega) = \frac{1}{\pi i} \oint_{\Gamma} \frac{\hat{s}(z)}{z-\omega}\, dz = \frac{1}{\pi i} \int_{\infty}^{-\infty} \frac{\hat{s}(z)}{z-\omega}\, dz + \frac{1}{\pi i} \int_{\text{half-circle}} \frac{\hat{s}(z)}{z-\omega}\, dz. \tag{2.110}$$

The fraction $1/\pi i$ appears, instead of the usual $1/2\pi i$, because the contour Γ slices exactly midway through the point $z = \omega$. Letting the loop on the lower half-plane expand to infinity, the last integral on the right vanishes, as the analytic function $\hat{s}(z)$ goes to zero exponentially fast. The remaining integral along the real axis can be rewritten using only real variables $z = \omega'$ and flipping the range of integration to obtain

$$\hat{s}(\omega) = \frac{1}{\pi i} \int_{-\infty}^{\infty} \frac{\hat{s}(\omega')}{\omega-\omega'}\, d\omega'. \tag{2.111}$$

Separating $\hat{s}(\omega)$ into real and imaginary parts, we obtain the following two equations:

$$\hat{s}_{\mathrm{R}}(\omega) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{\hat{s}_{\mathrm{I}}(\omega')}{\omega-\omega'}\, d\omega', \quad \hat{s}_{\mathrm{I}}(\omega) = -\frac{1}{\pi} \int_{-\infty}^{\infty} \frac{\hat{s}_{\mathrm{R}}(\omega')}{\omega-\omega'}\, d\omega', \tag{2.112}$$

which we recognize as the Hilbert transform from Eq. (2.99). The previous two equations can be written compactly as

$$\hat{s}_{\mathrm{R}}(\omega) = \mathcal{H}\left[\hat{s}_{\mathrm{I}}\right](\omega), \qquad \hat{s}_{\mathrm{I}}(\omega) = -\mathcal{H}\left[\hat{s}_{\mathrm{R}}\right](\omega). \tag{2.113}$$

Thus we conclude that a causal signal has the symmetry that the real and imaginary parts of its spectrum are not independent but rather are Hilbert transforms of one another.

Exercises

2.4.20 By direct calculation, verify that the causal signal $s(t) = e^{-t}$, for $t > 0$, has real and imaginary spectra given by

$$\hat{s}_{\mathrm{R}}(\omega) = \frac{1}{1+\omega^2} \text{ and } \hat{s}_{\mathrm{I}}(\omega) = \frac{-\omega}{1+\omega^2},$$

and that these two functions are Hilbert transforms of each other (keeping track of signs).

2.4.21 A causal signal $s(t)$ satisfies the equation

$$s(t) = h(t)s(t), \tag{2.114}$$

where $h(t)$ is the *Heaviside step function* or *unit causal function* defined by

$$h(t) = \begin{cases} 1, & t \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.115}$$

Their Fourier transforms must then satisfy the convolution relation $\hat{s} = \hat{s} \bullet \hat{h}$. Check that the Fourier transform of the Heaviside function is $\delta(\omega)/2 + 1/i\omega$, and verify that the convolution result from Eq. (2.114) gives the same Hilbert transform relation on the real and imaginary spectra $\hat{s}_{\mathrm{R}}(\omega)$ and $\hat{s}_{\mathrm{I}}(\omega)$.

2.4.22 In this section, we had a time-domain signal that vanished for $t < 0$, and we discovered that the real and imaginary parts of its spectrum were Hilbert transform pairs. In Section 2.4.6, we built a complex-valued signal in the time domain, where by construction the real and imaginary parts were Hilbert transform pairs, and showed that the spectrum vanished for $f < 0$. Construct a diagram of both of these situations and discuss the essential symmetry that induces a Hilbert pair relation. When this is done in the time domain, we showed that the Hilbert transform is a 90° phase rotation. Can we say the same about the Hilbert transform in the frequency domain?

2.4.8 Minimum Phase

In geoscience we often measure signals that come from an impulsive source, such as a dynamite blast, a weight drop, or even an earthquake. Such causal signals are characterized by the physical observation that most of their energy is concentrated near the start time $t = 0$, and are said to have *minimum phase delay*. Intuitively, the idea is that each frequency component of the impulse source has a phase that is as small as possible (the minimum phase delay) while leaving the total signal zero for times $t < 0$ (that is, causal).

This can be formalized into a definition of minimum phase for continuous-time signals:

Minimum phase A causal signal $s(t)$ is said to be *minimum phase* if it maximizes the energy in any interval $[0, T]$ compared with any other causal signal $r(t)$ with the same amplitude spectrum. That is,

$$\int_0^T |s(t)|^2\, dt \geq \int_0^T |r(t)|^2\, dt \text{ for any } T > 0, \text{ and any } r(t) \text{ with } |\hat{r}(f)| \equiv |\hat{s}(f)|.$$

This definition becomes a computational tool for finding minimum-phase signals by use of the following result:

Theorem *A causal signal $s(t)$ is minimum phase if and only if its extended Fourier transform $\hat{s}(z)$ on the lower half-plane is given as an integral of its log amplitude spectrum,*

$$\hat{s}(z) = \lambda \exp\left(\frac{1}{\pi}\int_{-\infty}^{\infty} \ln|\hat{s}(\omega)|\frac{\omega z - 1}{\omega - z}\frac{d\omega}{1+\omega^2}\right), \tag{2.116}$$

for all z in the lower half-plane, and some complex constant λ. In other words, its Fourier transform $\hat{s}(z)$ is an outer function in the sense of complex variables.

To verify this result, suppose a causal signal $r(t)$ has the same amplitude spectrum as the function s given by the above formula. Then the extended Fourier transform of r can be factored as

$$\hat{r}(z) = \hat{s}(z)\hat{g}(z), \text{ for all } z \text{ in the lower half-plane,} \tag{2.117}$$

where the function $\hat{g}(z) = \hat{r}(z)/\hat{s}(z)$ is analytic on the half-plane (since $\hat{s}(z)$ has no zeros, as it is an exponential), with constant magnitude 1 on the real line (since $|\hat{r}(f)| \equiv |\hat{s}(f)|$). This is called the inner/outer factorization of $\hat{r}(z)$, where $\hat{s}(z)$ is the outer function and $\hat{g}(z)$ is the inner function. Note that the function g is itself causal, since its Fourier transform $\hat{g}(z)$ is analytic on the half-plane.

With the use of $s_T(t)$, the truncation of $s(t)$ to the interval $[0, T]$ is defined as

$$s_T(t) = \begin{cases} s(t) & \text{if } t \in [0, T], \\ 0 & \text{otherwise,} \end{cases} \tag{2.118}$$

it is easy to check that $s \bullet g = s_T \bullet g$ on the interval $[0, T]$, since both functions are causal. It is also true that the energies agree, $\|s_T\| = \|s_T \bullet g\|$, since the amplitude spectra agree as shown by the equation

$$|\hat{s}_T(f)| = |\hat{s}_T(f)|\,|\hat{g}(f)| = |\widehat{(s_T \bullet g)}(f)|, \tag{2.119}$$

where the Fourier transform of g has a constant amplitude spectrum of 1. That is, the amplitude spectra of the two signals s_T and $s_T \bullet g$ agree, so they have the same energy.

Finally, when we compute energies on the time interval $[0, T]$, we see

$$\int_0^T |s(t)|^2\,dt = \|s_T\|^2 \tag{2.120}$$

$$= \|s_T \bullet g\|^2 \tag{2.121}$$

$$\geq \int_0^T |(s_T \bullet g)(t)|^2\,dt \tag{2.122}$$

$$= \int_0^T |(s \bullet g)(t)|^2\,dt \tag{2.123}$$

$$= \int_0^T |r(t)|^2\,dt. \tag{2.124}$$

That is, the partial energy of the minimum-phase signal $s(t)$ is greater than or equal to the partial energy of the original signal $r(t)$.[11]

There is another characterization of minimum phase that arises from the Hilbert transform. If a causal signal $s(t)$ is minimum phase, then its analytic Fourier transform $\hat{s}(z)$ is nonzero in the lower half-plane, and so its log spectrum $L(z) = \log(\hat{s}(z))$ is also analytic in the lower half-plane, and thus the corresponding signal is also causal. As noted in Section 2.4.7, this implies that the real and imaginary parts

$$L_{\mathrm{R}}(z) = \log|\hat{s}(z)| = \ln[A_s(z)], \qquad L_{\mathrm{I}}(z) = \arg(\hat{s}(z)) = \phi_s(z) \tag{2.125}$$

form Hilbert transform pairs. We can then write

$$\ln(A_s(\omega)) = \mathcal{H}\left[\phi_s(\omega)\right] \text{ and } \phi_s(\omega) = -\mathcal{H}\left[\ln(A_s(\omega))\right]. \tag{2.126}$$

In particular, this says that given the amplitude spectrum of a causal signal, we can compute its minimum-phase equivalent by computing its phase spectrum using the Hilbert transform. That is to say, the minimum-phase signal is recovered by exponentiating the log spectra, to obtain the Fourier transform of the constructed signal, as

$$\hat{s}(\omega) = \exp\left[\ln(A_s(\omega)) + i\phi_s(\omega)\right]. \tag{2.127}$$

We summarize this characterization as follows:

Theorem *If a causal signal $s(t)$ is minimum phase, then the log amplitude spectrum and phase spectrum form a Hilbert transform pair. That is to say, the real and imaginary parts of $\log\hat{s}(\omega)$ are Hilbert transforms of each other.*

In geophysical practice, we often have good estimates for amplitude spectra, but not phases, so this algorithm gives a practical method for computing phase spectra for minimum-phase signals.

[11] The astute reader will notice there is something unusual about the function $g(t)$, since its amplitude spectrum is a constant equal to 1, implying it has infinite energy. In fact, it can be shown that $g(t)$ is in the form of a signal plus its derivative, so although the signal has finite energy, its derivative does not. Nevertheless, the mathematical arguments above are valid.

We will see in Section 3.3.1 that, for discrete signals, minimum phase includes any stable, causal signal that has a causal stable inverse. Unfortunately, for continuous-time signals, there are never inverses for the convolution operation. That is because a signal $s(t)$ with finite energy and with inverse $s^{-1}(t)$ would satisfy the convolutional identity

$$s \bullet s^{-1} = \delta, \tag{2.128}$$

where δ is the Dirac delta. This is impossible, since the convolution of two functions is another function, while the Dirac delta is a distribution. Moreover, in the Fourier transform domain, we would have the equation

$$\hat{s}(\omega)\widehat{s^{-1}}(\omega) = 1, \tag{2.129}$$

which says that the product of two finite-energy functions becomes the constant function 1, which has infinite energy. Again, this is not possible.

So, we cannot use causal inverses to characterize a minimum-phase signal in continuous time.

An excellent summary of the mathematics of outer functions in complex analysis is contained in Hoffman (1962), from which these characterizations of minimum phase can be derived. Also from this mathematical theory, it is known that every causal signal $r(t)$ has a minimum-phase equivalent – that is, a function $s(t)$ with the same amplitude spectrum as $r(t)$ and satisfying the minimum-phase criteria. As well, any zero phase signal with finite time duration will have a minimum-phase equivalent, a fact that is routinely used in seismic deconvolution.

In summary, we have three useful characterizations of a minimum-phase signal: (i) it is a causal signal which has the energy maximized near the initial time $t = 0$; (ii) its Fourier spectrum extends to an outer function on the lower half-plane; and (iii) its log amplitude spectrum and phase spectrum are a Hilbert transform pair. The phase spectrum defined by Eq. (2.126) is given the special name of the minimum-phase spectrum. The physical intuition is that the minimum-phase signal is the most *front-loaded* signal possible that both is causal and has the given amplitude spectrum. Computationally, we can find the minimum-phase signal from its amplitude spectrum, using either the outer-function formula or the Hilbert transform pairs.

2.4.9 Computing Minimum Phase

From Eq. (2.126), the minimum-phase spectrum is computed by taking the negative Hilbert transform of the amplitude spectrum, giving the formula

$$\phi_s(\omega) = -\frac{1}{\pi}\int_{-\infty}^{\infty} \frac{\ln(A_s(\omega'))}{\omega - \omega'}\, d\omega'. \tag{2.130}$$

As stated, this integral is problematic because the log factor diverges to minus infinity as $\omega' \to \pm\infty$, simply because the amplitude spectrum $A_s(\omega')$ goes to zero as $\omega' \to \pm\infty$.

We can improve the formula by using the fact that for real signals, the amplitude spectrum is an even function, so we make a change of variables to show

$$\phi_s(\omega) = -\frac{1}{\pi}\left[\int_0^\infty \frac{\ln(A_s(\omega'))}{\omega - \omega'} + \frac{\ln(A_s(\omega'))}{\omega + \omega'}\, d\omega'\right] = -\frac{2\omega}{\pi}\left[\int_0^\infty \frac{\ln(A_s(\omega'))}{\omega^2 - (\omega')^2}\, d\omega'\right]. \tag{2.131}$$

It turns out that for causal signals with an amplitude spectrum $A_s(\omega)$, the denominator $\omega^2 - (\omega')^2$ goes to infinity fast enough as $\omega' \to \pm\infty$ to cancel out the growth of the logarithm $\ln(A_s(\omega'))$, so this integral converges. Thus, given the amplitude spectrum of any causal signal, we can compute the equivalent minimum phase from Eq. (2.131).

Not every signal has a minimum-phase equivalent. For instance, a Gaussian $s(t) = e^{-t^2}$ has a Fourier transform which is also a Gaussian, so the log spectrum $\ln(A_s(\omega))$ is proportional to ω^2. Thus the integrand above grows like $(\omega')^2/(\omega')^2 = 1$, which does not vanish at infinity – so the integral does not converge. However, all *causal* signals do have minimum-phase equivalents.

Exercises

2.4.23 Show that the Ricker wavelet has no exact minimum-phase equivalent, by showing that the defining integral for its minimum phase diverges. Nevertheless, in practice it is common to make a minimum-phase wavelet whose spectrum is similar to that of the Ricker wavelet.

2.5 Multidimensional Fourier Transforms

Seismic shot records are 3D constructs because it takes two spatial coordinates to describe the location of each receiver (assuming that the receivers are on the Earth's surface so that the vertical Cartesian coordinate is determined by the two horizontal coordinates), and time provides the third coordinate. When the source and the receivers are laid out in a straight line, then we have the special case of so-called 2D geometry, although the recorded wavefield is, of course, sill a 3D wavefield. A seismic dataset may have many thousands of source positions, each recorded separately into many thousands of receiver positions, and is a 5D construct (two dimensions for the receivers, two for the sources, plus time). For such a dataset, there are a great many *gathers* that can be analyzed independently. A gather is simply an ensemble of traces with some defining source–receiver geometry. For example, a source gather is simply all the traces recorded for a given source. A receiver gather is all of the traces recorded by a given receiver from any source. A common-offset gather is all of the traces for which the distance between source and receiver is a constant. A common-midpoint gather is all of the traces for which the midpoint of the line connecting source and receiver is constant. Depending upon circumstances, these gathers are usually either 2D or 3D. Given such a plethora of spatial coordinates and the corresponding trace

gathers, there is obviously a need for multidimensional data-processing algorithms, and the multidimensional Fourier transform is a fundamental tool. Here we will examine in some detail the 2D Fourier transform where one dimension is space and the other is time. This has become known as the *f–k* transform, where *f* is the familiar temporal frequency and *k* is a similar spatial frequency, or *wavenumber*. The physical units of *k* are cycles/meter (MKS system), to be compatible with *f* in cycles/second.

2.5.1 The Forward and Inverse *f–k* Transforms

Consider an arbitrary function of two variables, $u(t,x)$; then a reasonable 2D generalization of the 1D Fourier transform, as expressed by Eq. (2.15), is

$$\hat{u}(f,k) = \iint_{-\infty}^{\infty} u(t,x) e^{-2\pi i(ft+kx)}\, dt\, dx. \tag{2.132}$$

However, in the seismic context, we are transforming wavefields, not arbitrary functions. This means that we place a physical interpretation on these transformations, and in this case the Fourier kernel $g(t,x;f,k) = \exp(-2\pi i(ft+kx))$ has the interpretation of a traveling wave and the form of Eq. (2.132) is not preferred. This is because when both k and f are positive, $g(t,x;f,k)$ represents a wave traveling in the *negative* x direction. To see this, realize that a wave is defined by its wavefront, and a wavefront is a surface of constant phase. At some point t,x the phase of g is $\phi_0 = -2\pi(ft+kx)$, where ϕ_0 is a constant. If the time increases to $t+\Delta t$, then the wave travels to $x+\Delta x$, where $\phi_0 = -2\pi(f(t+\Delta t)+k(x+\Delta x))$. This can only be true if $f\Delta t + k\,\Delta x = 0$ or $\Delta x = -f\,\Delta t/k$, which is a negative number if both f and k are positive. It is preferable to use a transform for which the Fourier kernels are traveling waves moving in the $+x$ direction when f and k have the same sign and in the $-x$ direction when the signs are different. So we prefer the kernel $g(t,x;f,k) = \exp(-2\pi i(ft-kx))$ for the forward transform and its complex conjugate for the inverse transform.

A second point of concern about Eq. (2.132) is that, since we have as many as four spatial coordinates to describe a seismic dataset, then we have potentially four different wavenumbers. To distinguish these, we will generally use a subscript that matches the spatial coordinate, such as in k_x, k_y, k_z for the x,y,z wavenumbers. Thus we will prefer the kernel $g(t,x;f,k_x) = \exp(-2\pi i(ft-k_x x))$. In 3D, we will use a wavenumber vector $\vec{k} = (k_x,k_y,k_z)$ and a position vector $\vec{x} = (x,y,z)$ and write the 3D kernel as $g(t,\vec{x};f,\vec{k})\exp(-2\pi i(ft-\vec{k}\cdot\vec{x}))$, and similarly for 2D.

Finally, suppose we were to choose to use ω as the frequency rather than f. Then we might write a kernel like $g(t,x;\omega,\kappa_x) = \exp(-i(\omega t-\kappa_x x))$, where $\omega = 2\pi f$ and $\kappa_x = 2\pi k_x$. However, this is not commonly done, and κ_x can be difficult to distinguish from k_x anyway. Instead, the common practice is to use the same symbol k_x with both ω and f kernels, and the reader then must understand that when ω is used the wavenumbers are also in radians/meter. This is hugely important when writing code, since the difference between radians/meter and cycles/meter is a factor of 2π, which is a very large phase shift.

With these considerations in mind, we define the forward 2D *f–k* transform of a seismic wavefield recording, $\psi(t,x)$, as

$$\hat{\psi}(f,k_x) = \iint_{-\infty}^{\infty} \psi(t,x) e^{-2\pi i(ft-k_x x)}\, dt\, dx, \tag{2.133}$$

and the inverse *f–k* transform as

$$\psi(t,x) = \iint_{-\infty}^{\infty} \hat{\psi}(f,k_x) e^{2\pi i(ft-k_x x)}\, df\, dk_x. \tag{2.134}$$

When radian measures are used, the transform pair is

$$\hat{\psi}(\omega,k_x) = \iint_{-\infty}^{\infty} \psi(t,x) e^{-i(\omega t-k_x x)}\, dt\, dx, \tag{2.135}$$

and the inverse *f–k* transform is

$$\psi(t,x) = \frac{1}{4\pi^2} \iint_{-\infty}^{\infty} \hat{\psi}(\omega,k_x) e^{i(\omega t-k_x x)}\, d\omega\, dk_x. \tag{2.136}$$

It is common to use the term "*f–k* transform" as a generic reference to any 2D Fourier transform where one variable is a frequency and the other a wavenumber, regardless of which frequency variable is used and which spatial coordinate is used.

As with the 1D case, the signal $\psi(t,x)$ is real-valued but the spectrum $\hat{\psi}(f,k_x)$ is complex-valued. The arguments presented in Section 2.4.2 for the redundancy of the negative frequencies still apply, but the negative wavenumbers are not redundant.[12] If Eq. (2.133) is considered as an iterated integration where the t integration is done first and then the x integration, then the signal $\psi(t,x)$ is real-valued for the first integration but not for the second. The first integration gives us the data in the frequency-space domain, say $\hat{\psi}(f,x)$, which is complex-valued, and this is then input into the second integration. Thus, from a mathematical perspective, we need both positive and negative wavenumbers. From a physical perspective, we need both signs because waves traveling in the $+x$ direction will be represented in the $+f, +k_x$ quadrant of (f,k_x) space while those traveling in the $-x$ direction will be in the $+f, -k_x$ quadrant. The two quadrants $-f, +k_x$ and $-f, -k_x$ are ignored as redundant.

It is interesting to calculate the transform of an idealized traveling wave analytically. Suppose we have a perfect Dirac impulse for a wavelet and this is traveling in the $+x$ direction at constant velocity v. Then the traveltime–distance relation for this wave is $t = x/v$ and the wave itself is represented as $\psi_\delta(t,x) = \delta(t - x/v)$. This is a traveling wave because the delta function represents a Dirac impulse at the location where its argument vanishes, and that is where $t = x/v$. It can be visualized as a "knife edge" in (t,x) space. As a preliminary, notice that the properties of the delta function give the Fourier transform

[12] Alternatively, we could have the negative wavenumbers redundant, and then the negative frequencies would matter.

pair

$$\int_{-\infty}^{\infty} \delta(t - t_0) e^{-2\pi i f t}\, dt = e^{-2\pi i f t_0} \tag{2.137}$$

and

$$\delta(t - t_0) = \int_{-\infty}^{\infty} e^{-2\pi i f t_0} e^{2\pi i f t}\, df = \int_{-\infty}^{\infty} e^{2\pi i (t - t_0) f}\, df. \tag{2.138}$$

Then the *f–k* transform of $\psi_\delta(t, x) = \delta(t - x/v)$ is

$$\hat{\psi}_\delta(f, k_x) = \iint_{-\infty}^{\infty} \delta(t - x/v) e^{-2\pi i (ft - k_x x)}\, dt\, dx. \tag{2.139}$$

Using the delta function sifting rule (Eq. (2.26)) to perform the t integration gives

$$\hat{\psi}_\delta(f, k_x) = \int_{-\infty}^{\infty} e^{-2\pi i (f/v - k_x) x}\, dx. \tag{2.140}$$

Comparing this result with Eq. (2.138) leads to the conclusion

$$\hat{\psi}_\delta(f, k_x) = \delta(f/v - k_x). \tag{2.141}$$

So, we have shown that the *f–k* transform of $\delta(t - x/v)$ is $\delta(f/v - k_x)$, meaning that a linear event in (t, x) space transforms into a linear event in (f, k_x) space. Furthermore, the slopes of the events are reciprocals of each other. In a homogeneous acoustic medium with wave speed v, the event $\delta(t - x/v)$ can be formed by a plane wave propagating horizontally along the x axis in the $+x$ direction, so the angle between the wavefront and the x axis is 90°. This angle is called the emergence angle, and if a similar plane wave is propagating upward with an emergence angle of θ then it is represented by $\delta(t - x \sin\theta / v)$. Letting $v_{\text{app}} = v / \sin\theta$ be the *apparent velocity*, we then have the *f–k* transform pair

$$\delta(t - x \sin\theta / v) = \delta(t - x/v_{\text{app}}) \Leftrightarrow \delta(f/v_{\text{app}} - k_x) = \delta(f \sin\theta / v - k_x), \tag{2.142}$$

where $\Leftrightarrow$ denotes a transform pair. This event intercepts the x axis at $x = 0$, while $\delta(t - (x - x_0)/v_{\text{app}})$ has an intercept of x_0. In a similar fashion, the more general transform pair

$$\delta(t - (x - x_0)/v_{\text{app}}) \Leftrightarrow e^{2\pi i x_0 f} \delta(f/v_{\text{app}} - k_x) \tag{2.143}$$

can be derived. Thus the effect of the spatial shift is contained entirely in the phase, so that all events of the form $\delta(t - (x - x_0)/v_{\text{app}})$ have the same amplitude spectrum. This is a very useful fact because it means that an *f–k* filter can be designed to reject all events of a given apparent velocity regardless of their location. It can also be concluded that radial lines of the form $f/k_x = v_{\text{app}}$ are lines of constant apparent velocity.

Figure 2.17a shows a snapshot of an upward-traveling plane wave[13] in (x, z) space at some time t_0. The plane wave makes an angle of $\theta = 15°$ with the x axis, and this is called

[13] The event is displayed in 2D space, so the *plane* has degenerated to a *line*, but we still use the name "plane wave."

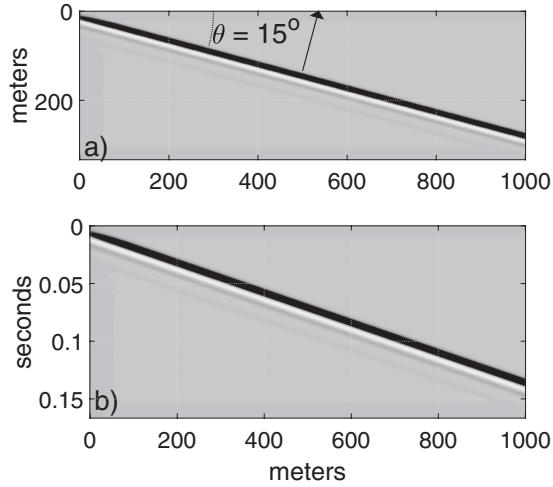


Figure 2.17a (a) A snapshot of an upward-traveling plane wave in (x, z) space at time $t = 0$, making an emergence angle of $\theta = 15°$. The normal to the plane wave is also shown. (b) The (t, x) space recording of the plane wave of panel a as recorded by receivers at $z = 0$. The time dip and spatial dip are related by $dt/dx = \sin\theta/v$.

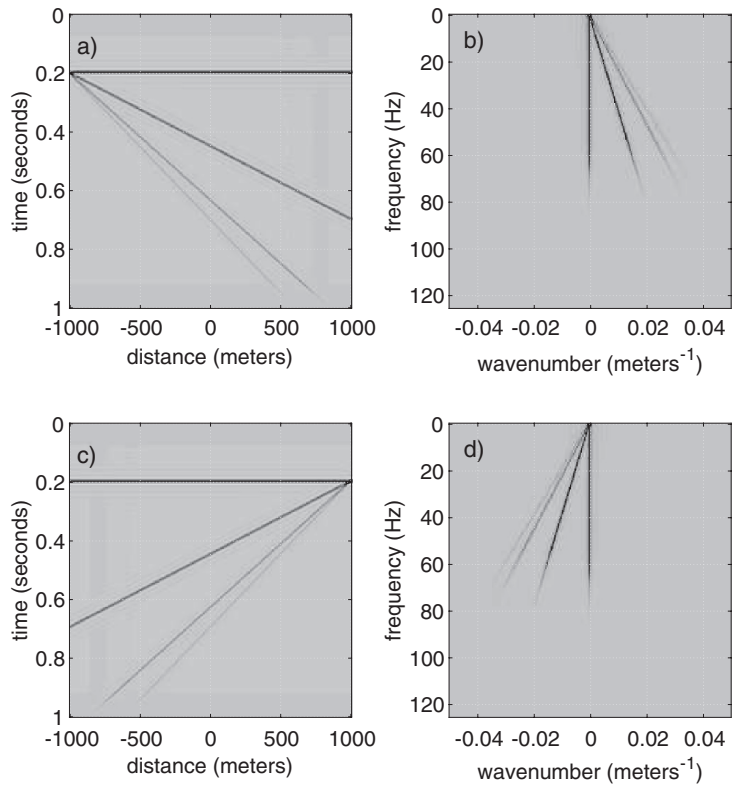


Figure 2.17b (a) A fan of four dipping events corresponding to apparent velocities $v_{app} = \sin\theta/v$ for $v = 2000$ m/s and $\theta = [0°, 30°, 60°, 90°]$. The events have amplitudes of 4, 3, 2, 1, respectively. (b) The *f*–*k* amplitude spectrum of the events in panel a. (c) Similar to panel a except that the events all have negative v_{app}. (d) The *f*–*k* amplitude spectrum of panel c.

the *emergence angle*. For receivers at $z = 0$, the wave will record as a linear event in (t, x) space with a traveltime $t = t_0 + x \sin\theta / v = t_0 + x / v_{\text{app}}$. Taking $t_0 = 0$ gives the picture in the lower panel of Figure 2.17a. The slope of the event in (t, x) space is called the *time dip*, while the actual dip of the wavefront in (x, z) space is the emergence angle. The time dip, which is also called the horizontal slowness, is calculated as the spatial derivative dt/dx and thus we have $dt/dx = \sin\theta / v$. Consider a fan of plane waves corresponding to a set of emergence angles ranging from 0° to 90°; the corresponding time dips will range between 0 and $1/v$. Therefore the maximum emergence angle (or event dip), which occurs for a vertical wavefront with $v_{\text{app}} = v$, gives a maximum time dip of $1/v$. So, while slopes range from horizontal to vertical in (x, z) space, they range from horizontal to $1/v$ in (t, x) space. Thus there is a maximum steepness for (t, x) space events. In the real world of variable velocity, this maximum dt/dx is usually determined by the slowest velocity, which occurs in the near surface.

Figure 2.17b, panel a, shows four events with emergence angles of 0°, 30°, 60°, and 90° in (t, x) space. As shown mathematically, these should transform to four linear events in (f, k_x) space, and the *f–k* amplitude spectrum is shown in panel b. These events have been assigned different amplitudes so that the reader can determine which *f–k* event corresponds to which *t–x* event. Inspection shows that the horizontal event in (t, x) space becomes vertical in (f, k_x) space. As shown mathematically, these events have slopes that are the inverse of one another. Thus the slowest apparent velocity causes the greatest slope in (t, x) space and the least slope in (f, k_x) space. This means that the *f–k* transform separates seismic events by their apparent velocity. Panels c and d of the same figure demonstrate that events with negative apparent velocity are found in the negative-wavenumber side of (f, k_x) space.

Code Snippet 2.5.1 illustrates the creation of the data displayed in Figure 2.17b. Two seismic sections (i.e., matrices) are created, each with four dipping events, where in the first section the events are down to the right and in the second they are down to the left. The *f–k* transform of both sections is accomplished with `fktran`. This function builds on MATLAB's FFT abilities but uses `fft` for the $t \rightarrow f$ transform and `ifft` for the $x \rightarrow k_x$ transform. This ensures that the Fourier kernel has different signs on the time and space terms. The inverse transform may be accomplished with `ifktran`. `fktran` requires the seismic section and its time and space coordinates as input and returns the complex-valued *f–k* transform together with its frequency and wavenumber coordinates. Only the positive frequencies are returned and, by default, the wavenumber axis is unwrapped (i.e., $k_x = 0$ is in the middle). The dipping events are created with `event_dip`, which inserts a simple linear event in a seismic matrix. This is only one of a set of similar commands that can insert linear, piecewise linear, or hyperbolic events. The option exists to create the events by hyperbolic superposition, which is useful in migration studies but will not be discussed further here. The command `help synsections` will give further information.

The simple transformation of linear events in (t, x) space to linear events in (f, k_x) space is not found for other event shapes. Generally speaking, the *f–k* mapping of an event is determined by the apparent velocities defined by $v_{\text{app}} = (dt/dx)^{-1}$, where $t(x)$ is the traveltime function defining the leading edge of the event. Conceptually, we imagine fitting tangent lines to $t(x)$ at each x and that information then maps to the radial line in (f, k_x)

Code Snippet 2.5.1 This example builds two simple seismic sections and computes their *f–k* transforms. Both sections have four dipping events, with emergence angles of 0°, 30°, 60°, and 90°(line 1), but in the first section the events are down to the right while in the second they are down to the left. The sections are initialized as all zeros (lines 5 and 6) and then the events are installed in a loop over angles (lines 7–16). Each event is installed by *event_dip* on line 11 for right dipping and line 14 for left dipping. This command simply adds a linear event to a matrix given the (t, x) coordinates of the ends of the event. After the loop, the sections are low-pass filtered (lines 17 and 18) and then *f–k* transformed (lines 19 and 20). The *f–k* transform is accomplished by *fktran*, which requires the seismic section and its time and space coordinates as input and returns the *f–k* transform and its frequency and wavenumber coordinates.

```
1   theta=0:30:90;v=2000;%dip and velocity
2   fmax=60;delfmax=20;%lowpass filter params
3   dt=.004;tmax=1;t=0:dt:tmax;%time coordinate
4   dx=10;xmax=1000;x=-xmax:dx:xmax;%x coordinate
5   seis1=zeros(length(t),length(x));%preallocate seismic matrix
6   seis2=seis1;%preallocate second seismic matrix
7   for k=1:length(theta)
8       t1=.2;%time at beginning of event
9       t2=t1+sind(theta(k))*(x(end)-x(1))/v;%time at end of event
10      %install event dipping to the right
11      seis1=event_dip(seis1,t,x,[t1 t2],[-xmax xmax],...
12          length(theta)-k+1);
13      %install event dipping to the left
14      seis2=event_dip(seis2,t,x,[t1 t2],[xmax -xmax],...
15          length(theta)-k+1);
16  end
17  seis1f=filtf(seis1,t,0,[fmax delfmax]);%lowpass filter on seis1
18  seis2f=filtf(seis2,t,0,[fmax delfmax]);%lowpass filter on seis2
19  [seis1fk,f,k]=fktran(seis1f,t,x);%fk transform of seis1
20  [seis2fk,f,k]=fktran(seis2f,t,x);%fk transform of seis2
```

End Code

signalcode/make_dip_fans.m

space given by $k_x/f = dt/dx = 1/v_{\text{app}}$. So, a linear event in *t–x* maps to a linear event in *f–k* because it has a constant value of dt/dx. Another class of events of great interest have hyperbolic traveltime curves of the form $t = \sqrt{t_0^2 + x^2/v^2}$, where t_0 is a constant. If x is the receiver coordinate (or source–receiver offset), this could describe a reflection event from a planar reflector. Alternatively, if x is the line coordinate on a stacked seismic section, this could describe a diffraction. Regardless, such event shapes are quite common. The traveltime gradient for such an event is $dt/dx = xv^{-2}/\sqrt{t_0^2 + x^2/v^2}$. At $x = 0$, we have $dt/dx = 0$, while for x very large, $dt/dx \to 1/v$. If the event extends for $x \in [-x_{\max} \to x_{\max}]$, in which $x_{\max}$ is very large, then it will have $dt/dx \in (-1/v \to 1/v)$. Figure 2.18a shows such an event as created by *event_hyp* and its *f–k* transform as computed by *fktran*. The significant part of the amplitude spectrum is confined to a triangular region with a

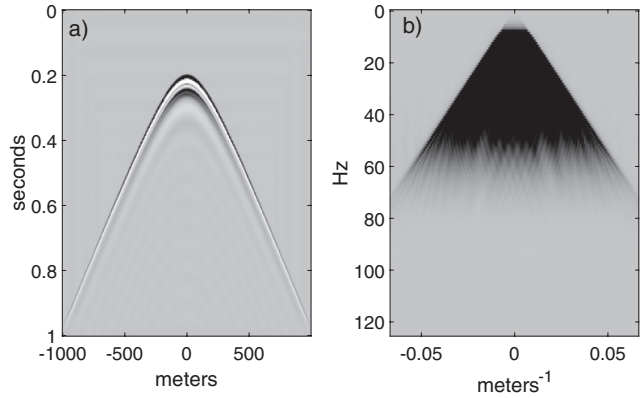


Figure 2.18a (a) A hyperbolic event in (t, x) space which might be either a diffraction or a reflection event. (b) The f–k amplitude spectrum of the seismic section displayed in panel a. Rather than being focused in (f, k_x) space, the hyperbolic event is spread across all apparent velocities corresponding to the range of tangents to its waveform. If v is the velocity defining the hyperbola, then these apparent velocities are in the range $v \to \infty$, corresponding to time dips of $1/v \to 0$.

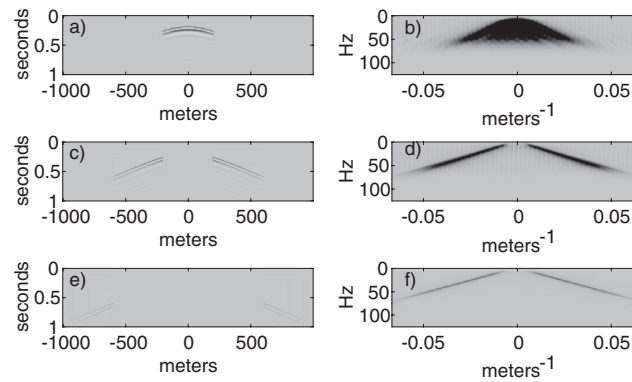


Figure 2.18b The f–k spectra of different portions of the hyperbolic event in Figure 2.18a, panel a, are shown. The central offsets are shown in (a) and (b), intermediate offsets in (c) and (d), and far offsets in (e) and (f). The portion of (f, k_x) space around $k_x = 0$ contains the most important central part.

slight apex truncation. This triangular region is actually determined by the wave nature of seismic data and is called the *wavelike region*[14] of (f, k_x) space. The hyperbolic event has a 10–60 Hz minimum-phase wavelet, which accounts for the upper and lower boundaries of this region, while the sides correspond to $dt/dx = \pm 1/v$ (here $v = 2100$ m/s). Clearly, the hyperbolic event has been spread across the entire range of expected time dips (or apparent velocities). For further understanding, Figure 2.18b shows the f–k spectra of three different offset ranges, where the offset is measured from the apex of the event. The

[14] Technically, the wavelike region has only the lateral bounds and not the upper and lower bounds that are seen here.

central one-third of the offsets are from near the apex, where the time dips are low and the apparent velocities are nearly infinite. This energy forms the central portion of the spectrum around $k_x = 0$. Since this is clearly the most important portion of the event, this region near $k_x = 0$ is often called the *data region* of (f, k_x) space. Also, after traveltime correction by *normal moveout removal*, the entire event will be linear with $dt/dx = 0$ and hence exactly at $k_x = 0$. The middle third of the offsets and the far third form portions progressively more distant from $k_x = 0$. While the central region is the most important, imaging theory requires that as much as possible of this wavelike region be recorded and properly processed into the final seismic image.

Exercises

2.5.1 Derive Eq. (2.143).

2.5.2 Write a code similar to that in Code Snippet 2.5.1 to create a seismic section with four linear events having identical apparent velocities but different *t–x* positions. Compute the *f–k* transform of this section and display its amplitude spectrum. Show that the amplitude spectrum is dominated by a single linear event with the apparent velocity of your choice. Describe how an *f–k* filter process could remove all of these events at once.

2.5.3 Study Figure 2.18b carefully. The (t, x) space wavefields are related to the wavefield of Figure 2.18a by a spatial window. Describe the window required for panel a and identify its imprint in the spectrum of panel b.

2.5.2 Solving the Wave Equation with the *f–k* Transform

The *f–k* transform can be used to develop an exact solution to the scalar wave equation when the velocity is constant. Despite the apparent shortcomings of the assumptions of constant velocity and scalar waves, there are some fundamental features that arise from this solution that are common to all types of waves in much more complicated media. Also, the history of seismic data processing shows that this solution will have application in a great many useful algorithms even when the media are inhomogeneous and the waves nonscalar. One very fundamental fact is that the physics of wave propagation causes the temporal and spatial bandwidths to be linked. This means that a signal that is somehow band limited in frequency is also necessarily band limited in wavenumber.

Let $\psi(t, x, z)$ be the wavefield that satisfies

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial z^2} - \frac{1}{v^2} \frac{\partial^2 \psi}{\partial t^2} = 0, \tag{2.144}$$

where v is the wave velocity, which is assumed to be constant. As written with the right-hand side equal to zero, this is a source-free expression, so the solution will be nontrivial only if the initial or boundary conditions are also nontrivial. The construction of a general solution to a constant-coefficient partial differential equation like this can usually be done

with Fourier techniques, where all independent variables except one are Fourier transformed. The z axis will be assumed vertical and to increase in the downward direction, while x will increase to the right. It is convenient to transform over t and x while retaining the z dependence because seismic data is (conceptually) collected on a plane of constant z as a function of t and x. The wavefield can then be written as an inverse f–k transform over its f–k spectrum $\hat{\psi}(f, k_x, z)$ as

$$\psi(t, x, z) = \iint_{-\infty}^{\infty} \hat{\psi}(f, k_x, z) e^{2\pi i(ft - k_x x)}\, df\, dk_x. \tag{2.145}$$

Substitution of Eq. (2.145) into Eq. (2.144) looks more daunting than it actually is. Consider the action of $\partial_x^2 \psi$ on Eq. (2.145),

$$\begin{aligned} &\frac{\partial^2}{\partial x^2} \iint_{-\infty}^{\infty} \hat{\psi}(f, k_x, z) e^{2\pi i(ft - k_x x)}\, df\, dk_x \\ &\qquad = \iint_{-\infty}^{\infty} \hat{\psi}(f, k_x, z) \underbrace{(-2\pi i k_x)^2}_{\text{action of } \partial_x^2} e^{i2\pi(ft - k_x x)}\, df\, dk_x. \end{aligned} \tag{2.146}$$

This happens because the x dependence in Eq. (2.145) occurs entirely in the Fourier kernel and exponentials are extremely simple to differentiate. Recall that $\partial_x e^{ax} = a e^{ax}$, so each x derivative simply causes a factor of $-2\pi i k_x$ to be introduced into the spectrum (i.e., the integrand) of the inverse transform. This is one of the great virtues of the Fourier transform, that it converts differentiation to multiplication. The first derivative can therefore be viewed as a filter with spectrum $-2\pi i k_x = 2\pi k_x e^{-\pi i/2}$, which has an amplitude spectrum of $2\pi k_x$ and a phase spectrum of $-\pi/2$. Thus a first derivative applies a linear ramp to the amplitude spectrum and a 90° phase rotation. The spectrum of the second derivative is $-4\pi^2 k_x^2 = 4\pi^2 k_x^2 e^{\pi i}$, which is a quadratic ramp and a 180° phase shift. In similar fashion, the spectral action of ∂_t^2 is $-4\pi^2 f^2$. The z derivative will remain and will operate on $\hat{\psi}(f, k_x, z)$.

Putting this together, it follows that the result of substitution of Eq. (2.145) into Eq. (2.144) is

$$\iint_{-\infty}^{\infty} \left[\left(-4\pi^2 k_x^2 + 4\pi^2 \frac{f^2}{v^2} \right) \hat{\psi}(f, k_x, z) + \frac{\partial^2 \hat{\psi}}{\partial z^2}(f, k_x, z) \right] e^{2\pi i(ft - k_x x)}\, df\, dk_x = 0. \tag{2.147}$$

This says that the inverse f–k transform of the term in square brackets must vanish. It can be shown that the uniqueness of the Fourier transform means that the zero signal (i.e., the right-hand side of this equation) must have a zero spectrum. Thus it follows that the term in square brackets must vanish for every f and k_x, which is expressed by

$$\frac{\partial^2 \hat{\psi}}{\partial z^2}(f, k_x, z) = -4\pi^2 \left(\frac{f^2}{v^2} - k_x^2 \right) \hat{\psi}(f, k_x, z). \tag{2.148}$$

So, by using the *f–k* transform we have reduced the partial differential equation of Eq. (2.144) to the ordinary differential equation of Eq. (2.148).[15] Actually, this is a family of ordinary differential equations for all possible values of f and k_x. The general solution of a second-order ordinary differential equation like this can be formed as a linear combination of any two independent particular solutions. It is convenient to define the constant k_z^2 by

$$k_z^2 = \frac{f^2}{v^2} - k_x^2 \tag{2.149}$$

and then rewrite Eq. (2.148) as

$$\frac{\partial^2 \hat{\psi}}{\partial z^2}(f, k_x, z) = -4\pi^2 k_z^2 \hat{\psi}(f, k_x, z). \tag{2.150}$$

This equation is often referred to as the Fourier-transformed wave equation. Taking k_z to indicate the positive square root of k_z^2, then two particular solutions to Eq. (2.150) are $\psi_+ = e^{i2\pi k_z z}$ and $\psi_- = e^{-i2\pi k_z z}$. (To see this, just plug either solution into the equation and see that an identity results.) We can then write the general solution to Eq. (2.150) as

$$\hat{\psi}(f, k_x, z) = A(f, k_x) e^{i2\pi k_z z} + B(f, kx) e^{-i2\pi k_z z}, \tag{2.151}$$

where $A(f, k_x)$ and $B(f, kx)$ are constants for any particular values of f and k_x that must be determined by boundary conditions, which are presumably prescribed on the z plane where the data are collected, which we will take to be $z = 0$. When this general solution is substituted into the inverse *f–k* transform machinery of Eq. (2.145), the exponentials $e^{\pm i2\pi k_z z}$ can be combined with those of the Fourier kernel and we can write the solution in (t, x, z) space as

$$\psi(t, x, z) = \iint_{-\infty}^{\infty} \left[\underbrace{A(f, k_x) e^{i2\pi(ft - k_x x + k_z z)}}_{\text{upward-traveling waves}} + \underbrace{B(f, k_x) e^{i2\pi(ft - k_x x - k_z z)}}_{\text{downward-traveling waves}} \right] df dk_x. \tag{2.152}$$

The upward-traveling wave term is identified by its Fourier exponential, in which the t and z terms have same signs. This means that the term is a wave traveling in the negative z direction.[16] In like fashion, the downward-traveling wave term is identified by opposite signs on the t and z terms. It can also be concluded that $A(f, k_x)$ is the *f–k* transform of upward-traveling waves collected on $z = 0$, while $B(f, k_x)$ is a similar spectrum of downward-traveling waves. This general solution is a rederivation of what is called the *d'Alembert solution* to the wave equation. Some 400 years ago, d'Alembert was the first to show that the wave equation could be solved quite generally as the superposition of two wavefields moving in opposite directions.

Most important for the present discussion is the meaning of $k_z = \sqrt{f^2/v^2 - k_x^2}$ when used in Eq. (2.151). The presence of a minus sign beneath the square root means that k_z

[15] We continue to use the partial derivative symbol as a reminder that f and k_x are held constant in the solution.
[16] If you don't know why, see the discussion on page 91.

can be either real or imaginary. When $f^2/v^2 > k_x^2$, k_z is real and $e^{\pm 2\pi i k_z z}$ is a complex exponential, representing traveling-wave behavior. Alternatively, when $k_x^2 > f^2/v^2$, k_z is purely imaginary (i.e., it has no real part) and $e^{\pm 2\pi i k_z z} = e^{\mp 2\pi |k_z| z}$, which represents decaying or growing real exponential behavior, meaning they are not traveling waves but *evanescent*[17] waves. Thus, we are compelled to divide (f, k_x) space into a *wavelike region* and an *evanescent region* (the nonwavelike part) according to

$$k_z = \begin{cases} \sqrt{\dfrac{f^2}{v^2} - k_x^2}, & \left|\dfrac{f}{v}\right| \geq k_x \quad \text{wavelike region,} \\ i\sqrt{k_x^2 - \dfrac{f^2}{v^2}}, & k_x > \left|\dfrac{f}{v}\right| \quad \text{evanescent region.} \end{cases} \tag{2.153}$$

When this more detailed definition of k_z is used in Eq. (2.151), it is important to choose the signs such that when k_z becomes evanescent only decaying, not growing, exponential behavior is allowed. A simple way to do this is

$$\hat{\psi}(f, k_x, z) = A(f, k_x) e^{2\pi i k_z z} + B(f, kx) e^{-2\pi i k_z^* z}, \tag{2.154}$$

where the complex conjugate of k_z, denoted by an asterisk, is used in the second term.

The separation of (f, k_x) space into wavelike and evanescent regions is depicted in Figure 2.19a. The evanescent boundary is defined by $|f/k_x| = v$ and in 2D is the two radial lines in the figure. In 3D, the boundary is a cone defined by $f/\sqrt{k_x^2 + k_y^2} = v$. Inside this boundary we have wavelike behavior, and these spectral points are useful in imaging. Outside the boundary is the evanescent region, which, in real data, is filled mainly with noise and is not generally useful. So, to any particular frequency $f > 0$ there corresponds a maximum wavelike wavenumber given by $|k_{\max}| = f/v$. This means that if there exists some mechanism that results in a maximum signal frequency $f_{\max}$, then this will also impose a maximum wavenumber $k_{\max} = f_{\max}/v$. When the seismic source is a Vibroseis vehicle emitting a programmed sweep, the highest frequency of the sweep is an estimate of $f_{\max}$. If the seismic source is impulsive, such as dynamite, then such explosions are known to produce a spectrum with a dominant frequency and a strong decay of higher frequencies. This, together with the Earth's natural anelastic attenuation, means that there will always be some higher frequency at which the signal has become so weak that it falls below the noise. In this context, $f_{\max}$ will be more difficult to estimate than for the Vibroseis case but it is still reasonable to suppose that it exists. Thus, the physics of wave propagation imposes a natural wavenumber band limit whenever a frequency band limit exists. The *f–k* spectra of hyperbolic events, as shown in Figure 2.18a, give a natural expression of this effect. In the real Earth, velocity tends to increase with depth, which means that $k_{\max}(z) = f/v(z)$ will decrease with depth. As a wavefield propagates down through a layered medium, its wavenumber bandwidth constantly shrinks, and upon reflection at some deep boundary, this bandwidth is not recovered on the upgoing path. Thus the wavenumber bandwidth available to illuminate a given target is determined by the highest velocity encountered in the wavepath.

[17] *Evanescent* means "quickly vanishing" or "rapidly decaying."

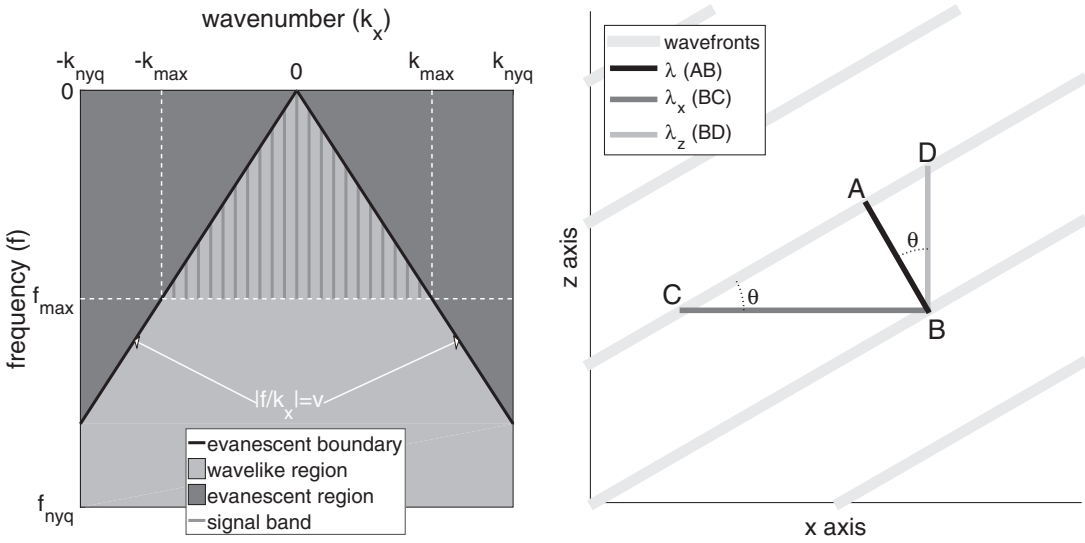


Figure 2.19a (left) A diagram of (f, k_x) space showing the separation into wavelike and evanescent regions. To some maximum signal frequency f_{max} there corresponds a maximum wavenumber in the wavelike region, $k_{max} = f_{max}/v$, such that the signal band is limited in both frequency and wavenumber. A frequency band limit therefore causes a wavenumber band limit.

Figure 2.19b (right) The wavefronts of a monochromatic plane wave make an angle θ with the x axis. The wavelength λ is the perpendicular distance between wavefronts, AB; the horizontal wavelength λ_x is the distance between wavefronts in the x direction, CB; and the vertical wavelength λ_z is BD. It follows that $\lambda_x = \lambda / \sin\theta$ and $\lambda_z = \lambda / \cos\theta$, and therefore $\lambda^{-2} = \lambda_x^{-2} + \lambda_z^{-2}$.

To better understand this effect and to realize why it applies to all waves and not just scalar waves, consider again Eq. (2.149), which can be rearranged as

$$\frac{f^2}{v^2} = k_x^2 + k_z^2. \tag{2.155}$$

This is often called the *dispersion relation for scalar waves* and is sometimes said to be the "wave equation in the Fourier domain." Indeed, just plug $\psi = e^{2\pi i(ft - k_x x - k_z z)}$ into the wave equation (Eq. (2.144)) and this result quickly emerges. It is a statement that the three Fourier variables (f, k_x, k_z) are not independent for waves. Any two can be specified, but then the wave equation determines the third. In seismic exploration, we collect data on a plane of constant z and, after an f–k transform, we have data as a function of f and k_x. This then determines k_z and the data at any z by an equation like Eq. (2.152) or similar. From basic physics, we know that for any wave $f\lambda = v$, where λ is the wavelength. So, the left-hand side of Eq. (2.155) is actually λ^{-2}, which suggests the identification of wavelength components $\lambda_x = k_x^{-1}$ and $\lambda_z = k_z^{-1}$ and thus the conclusion $\lambda^{-2} = \lambda_x^{-2} + \lambda_z^{-2}$. Figure 2.19b shows a geometric interpretation of this result. For a monochromatic plane wave, λ is the distance between wavefronts measured perpendicular to the fronts. The wavelength components λ_x and λ_z are the distances between wavefronts measured along the x and z coordinate axes, respectively. So, we see that the "components" of wavelength

are always greater than the wavelength itself and that it is not possible to consider wavelength as a vector quantity. However, the inverse of wavelength, which is the wavenumber, is naturally a vector that points in the direction of wave propagation and has magnitude $k = \sqrt{k_x^2 + k_z^2}$. In fact, using $\vec{k} = k_x\hat{x} + k_z\hat{z}$ ($\hat{x}$ and $\hat{z}$ are unit vectors in the coordinate directions) and a similar position vector $\vec{r} = x\hat{x} + z\hat{z}$ allows the Fourier kernel $e^{2\pi i(ft - k_x x - k_z z)}$ to be written $e^{2\pi i(ft - \vec{k}\cdot\vec{r})}$, which has a direct generalization to 3D. So the dispersion relation for scalar waves, which links the frequency and wavenumber bandwidths, is a direct consequence of the geometry of monochromatic plane waves and will always apply, perhaps in a space-variant form in heterogeneous media, whenever the multidimensional Fourier transform is used.

2.6 Chapter Summary

Here we have introduced the fundamental concepts of signal processing and analysis in the setting of continuous signals, or simply mathematical functions with finite energy. We defined correlation and convolution and showed their relationship and their various properties. We then defined the Fourier transform and explored its properties, especially its link with convolution. Amplitude and phase spectra were defined and the important concept of minimum phase was introduced. Finally, we defined and explored the Fourier transform in multiple dimensions. Throughout this chapter, we have presented figures illustrating these concepts that were computed from sampled signals. Sampling is the subject of the next chapter.

3 Signal Theory: Discrete

3.1 Sampling

In the previous chapter, the discussion has been about continuous-time signals, meaning signals that are defined for all possible values of time. Usually this means we have an equation for the signals and, if we want to operate on them, perhaps convolve two signals together or examine a signal's spectrum, then we must use the tools of calculus and calculate one or more integrals. This has allowed us to build a powerful theory for such signals and has led to some significant insights. Throughout the discussion, there has been occasional reference to samples and, in fact, all of the numerical examples presented in that chapter have used sampled signals and appropriate algorithms. We now turn to a systematic study of sampling and its implications.

In order to deal with real data, we must acknowledge that such formulas can only describe signals in the abstract sense and that real seismic traces are represented by samples, not formulas. We must extend our theory to deal with such sampled signals. Moreover, our operations on these signals will almost always be done in a computer, so we must develop numerical computation methods that manipulate sampled signals in both the time and the frequency domains. We might expect that our integrals for convolution or Fourier transformation will become summations, because in calculus an integral is just the limiting form of a discrete sum. However, that is just the beginning; there is much more to the story than that. Of first importance is to understand the mathematical basis for sampling, meaning that we need to know when a signal can be represented by samples without loss of information. The fundamental result is called the *sampling theorem* and is credited to C. E. Shannon, H. Nyquist, E. T. Whittaker, and V. A. Kotelnikov, who all made essentially independent discoveries of this important result in the early part of the twentieth century. The most straightforward application of the sampling theorem is in properly sampling an electrical signal, such as that from a geophone, in the time domain. Then, in order to understand the Fourier transform of sampled signals, we must apply the sampling theorem in the frequency domain, and this will lead us to the *discrete Fourier transform*. Having built a theory for sampled signals, we will find that there are still times when we wish to recover the underlying continuous signal from its samples. This is the process of interpolation. A closely related concern is to develop methods to change the sample interval from smaller to larger or the reverse.

The sampling of a continuous signal $s(t)$ refers to a process of selecting a finite, discrete set of values $[s(t_1), s(t_2), s(t_3), \ldots, s(t_N)]$, where the sample times $t_1, t_2, \ldots, t_N$ are somehow chosen to avoid loss of information. At first thought, it might seem that it is

impossible to avoid losing information, even if $s(t)$ has finite length, because there are infinitely many points in any finite interval of the real line (i.e., between any two times t_k and t_j). However, this is possible if the signal is band limited, meaning that $\hat{s}(f)$ vanishes for any frequency greater than some maximum $f_{\max}$. In fact, the *sampling theorem* tells us that to avoid information loss, we must take our samples at regular intervals Δt, where $\Delta t < (2f_{\max})^{-1}$. Commonly, seismic data is sampled at one of the standard sample intervals $\Delta t = [0.0005, 0.001, 0.002, 0.004, 0.008]$, measured in seconds. Corresponding to each sample interval is the *Nyquist frequency* $f_{\text{nyq}} = 1/(2\,\Delta t)$, with values of $f_{\text{nyq}} = [1000, 500, 250, 125, 62.5]$ Hz. So, if data is acquired with a band-limited source, perhaps a Vibroseis machine, such that the maximum frequency is $f_{\max}$, then a sample interval is chosen from the standard ones such that the corresponding Nyquist frequency satisfies $f_{\max} < f_{\text{nyq}}$. With modern instrumentation, samples are relatively cheap and it is not uncommon to choose to oversample by a factor of 2 so that $f_{\max} < f_{\text{nyq}}/2$. If the source has no definite band limit, for example in the case of dynamite, then the sample interval is chosen such that the corresponding Nyquist frequency generously allows sufficient bandwidth for the purpose of exploration. Then, prior to sampling, the data is forced to be band limited by passing it through an *antialias filter*, to be explained shortly.

3.1.1 Sampling and Reconstruction

In order to store and manipulate a signal $s(t)$ in the computer, typically we sample the signal on a uniform grid of time steps $t = 0, \pm 1\,\Delta t, \pm 2\,\Delta t, \pm 3\,\Delta t, \ldots$. This produces a sequence of real or complex numbers

$$s_k = s(k\,\Delta t), \qquad k = 0, \pm 1, \pm 2, \pm 3, \ldots, \tag{3.1}$$

and only a finite number of nonzero coefficients are stored, since computer memory is limited. This is not an exact representation of the original function $s(t)$, but for Δt chosen sufficiently small, a close approximation to $s(t)$ can be reconstructed from the sequence of values $[s_k]$. Such a sequence $[s_k]$ is called a sampled, or *discrete-time*, signal, to distinguish it from the continuous-time signal $s(t)$.[1]

A piecewise constant approximation to a smooth signal $s(t)$ can be constructed as shown in Figure 3.1a, obtained by forming a linear combination of boxcar functions

$$s_{\text{pc}}(t) = \sum_k s_k \cdot \chi\left(\frac{t - k\,\Delta t}{\Delta t}\right). \tag{3.2}$$

Here, the symbol $\chi(t)$ denotes the unit boxcar function that is equal to one on the interval $[-0.5, 0.5]$ and zero everywhere else. The shift $t - k\,\Delta t$ in the argument of the boxcar function centers it at the sample point $t_k = k\,\Delta t$, and the weight s_k scales the boxcar so that it reaches the sample height.

[1] The word "continuous" here refers to the fact that the parameter t varies continuously over the real line, even though the function $s(t)$ might not actually be continuous.

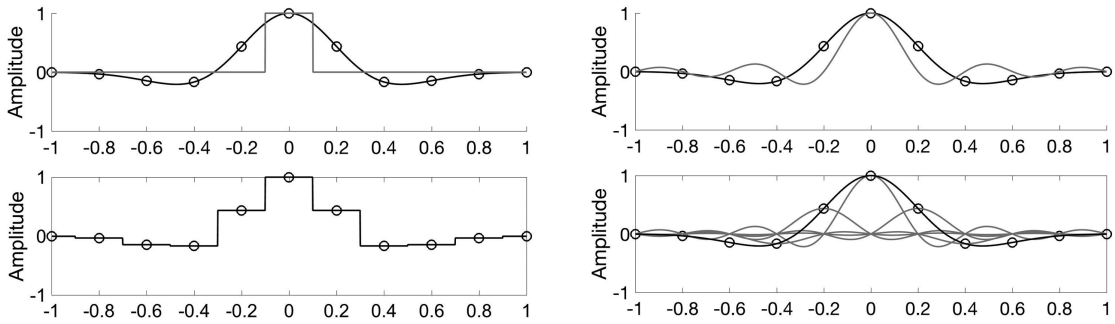


Figure 3.1a (left) A smooth bump function, sampled at time intervals in multiples of 0.2. The boxcar in the center is a very rough approximation. The piecewise constant approximation shown on the bottom is computed by adding translates of the boxcar function, scaled by the height at each sample point.

Figure 3.1b (right) The same bump function and a sinc function used for an approximation. The smooth approximation shown on the bottom is computed by adding translates of the sinc function (in gray), scaled by the height at each sample point. The result (in black) is an excellent approximation to the original.

A smooth approximation is obtained by replacing the boxcar $\chi(t)$ with a smooth function such as a spline, as shown in Figure 3.1b. It is very convenient to use a sinc function[2] to obtain a formula for a smooth approximation as

$$s_{\mathrm{sm}}(t) = \sum_k s_k \cdot \mathrm{sinc}\left(\frac{t - k\,\Delta t}{\Delta t}\right). \tag{3.3}$$

Notice that since the sinc function is zero on all the integers, except at $t = 0$, this smooth approximation is exact at the sample points $j\,\Delta t$, for the sum then collapses as

$$s_{\mathrm{sm}}(j\,\Delta t) = \sum_k s_k \cdot \mathrm{sinc}\left(\frac{(j-k)\,\Delta t}{\Delta t}\right) = s_j = s(j\,\Delta t). \tag{3.4}$$

This smooth approximation is in fact a band-limited signal, which we hope will be close to the original signal $s(t)$. We can examine the frequency content of $s_{\mathrm{sm}}(t)$ by taking its Fourier transform. The sinc function will transform to a boxcar function of width $1/\Delta t$ in the frequency domain.[3] Translates of the sinc function will give the same boxcar, modulated by the complex exponential $e^{-2\pi ikf\,\Delta t}$. Thus we have the Fourier transform expressed as the sum

$$\hat{s}_{\mathrm{sm}}(f) = \Delta t \sum_k s_k \cdot \chi(f\Delta t) e^{-2\pi ikf\,\Delta t}, \tag{3.5}$$

or, in the more familiar form,

$$\hat{s}_{\mathrm{sm}}(f) = \Delta t \sum_k s_k \cdot e^{-2\pi ikf\,\Delta t} \quad \text{for} \quad -\frac{1}{2\,\Delta t} < f < \frac{1}{2\,\Delta t} \tag{3.6}$$

and zero otherwise.

[2] The normalized sinc function is defined as $\mathrm{sinc}(t) = \frac{\sin(\pi t)}{\pi t}$.

[3] It is easy to see this by computing the inverse Fourier transform of the boxcar function. See (3.28) for details.

The function $s_{\mathrm{sm}}(t)$ is an example of a band-limited signal, where its frequency content is restricted to frequencies in the interval $[-1/(2\,\Delta t), 1/(2\,\Delta t)]$. The original signal $s(t)$ might not be band limited, in which case the two signals $s(t)$ and $s_{\mathrm{sm}}(t)$ must be different, since their Fourier transforms are different.

When the signal $s(t)$ is band limited, with its Fourier transform supported in the same interval $[-1/(2\,\Delta t), 1/(2\,\Delta t)]$ as $s_{\mathrm{sm}}(t)$, the Shannon sampling theorem says that in fact these two signals are equal to each other, for all times t. The intuition for this powerful result is as follows. The difference function $d(f) = s(t) - s_{\mathrm{sm}}(t)$ is equal to zero at all sample points $\pm k\,\Delta t$, so if it was nonzero anywhere, it would have to be nonzero in the interior of some interval $[k\,\Delta t, (k+1)\,\Delta t]$. The function must then oscillate from zero at $k\,\Delta t$ to a nonzero value, then back to zero at $(k+1)\,\Delta t$, indicating an oscillation with a period less than $2\,\Delta t$ (i.e., we can fit at least half a cycle into the interval). Its frequency must then be greater than $1/(2\,\Delta t)$, contradicting the fact that $s(t)$, $s_{\mathrm{sm}}(t)$, and $d(t)$ all have Fourier transforms that are nonzero only in the interval $[-1/(2\,\Delta t), 1/(2\,\Delta t)]$.

Given any band-limited signal $s(t)$, we can find a sample rate such that the samples s_k can be used to form an exact reconstruction of the original signals, as follows. Choose a frequency $f_{\max}$ large enough so that the Fourier transform $\hat{s}(f)$ is supported in the interval $[-f_{\max}, f_{\max}]$.[4] Set the sampling interval to be $\Delta t = 2f_{\max}$. By the discussion above, we know that the smooth approximation $s_{\mathrm{sm}}(t)$ will also be band limited, supported in the same interval $[-1/(2\,\Delta t), 1/(2\,\Delta t)] = [-f_{\max}, f_{\max}]$. By the Shannon sampling theorem, we conclude that the smooth approximation $s_{\mathrm{sm}}(t)$ reconstructed from samples of $s(t)$ is exactly equal to the original signal.

The reciprocal $1/\Delta t$ is called the sample rate, while half of it, $1/(2\,\Delta t)$, is called the Nyquist rate. A quick summary of the Shannon reconstruction result is that the Nyquist rate must be greater than the highest frequency present in the signal $s(t)$ in order to completely capture the information in the signal through samples s_k.

In practice, we usually choose the sample rate to be significantly greater than twice the sample rate. For instance, in land seismic surveys, we expect geophone signals in the range of 0 to 200 Hz. This suggests that the minimum sample rate is 400 samples per second, yet typically we use 500 or 1000 samples per second for sampling. For music stored in computers and digital recordings on CDs and DVDs, the sample rate is typically 44 100 sample per second, about 10% higher than twice the 20 kHz frequency limit for human hearing.

When a signal is sampled at too slow a rate, we get the phenomenon of aliasing, where frequencies that are higher than the Nyquist rate are recorded as if they were lower frequencies. To avoid this problem, the raw signal is usually conditioned with a low-pass analog filter to remove those higher frequencies.

3.1.2 Discrete Convolution, Correlation, Energy

The Shannon sampling theorem tells us that a band-limited signal $s(t)$ can be reconstructed exactly from its samples $s_k = s(k\,\Delta t)$ using Eq. (3.3) provided the sample rate $1/\Delta t$ is

[4] We can set $f_{\max}$ to be the maximum frequency present in the signal $s(t)$.

greater than twice the highest frequency in the signal. Remarkably, these discrete samples also allow us to compute the convolutions, correlation, and energy for continuous-time signals as well.

We begin with convolution. Suppose $r(t), s(t)$ are band-limited signals and $c(t) = (r \bullet s)(t)$ is its convolution. Let r_j, s_k, c_n be the corresponding samples of the functions. Under the Fourier transform, the convolution becomes a product, so $\hat{c}(f) = \hat{r}(f)\hat{s}(f)$ and we conclude that the signal $c(t)$ is also band limited. We use Eq. (3.6) to express each Fourier transform as a sum of exponentials on the interval $[-1/(2\,\Delta t), 1/(2\,\Delta t)]$, to see that

$$\hat{r}(f) = \Delta t \sum_j r_j \cdot e^{-2\pi ijf\Delta t}, \tag{3.7}$$

$$\hat{s}(f) = \Delta t \sum_k s_k \cdot e^{-2\pi ikf\Delta t}, \tag{3.8}$$

$$\hat{c}(f) = \Delta t \sum_n c_n \cdot e^{-2\pi inf\Delta t}. \tag{3.9}$$

Equating $\hat{c}(f) = \hat{s}(f)\hat{r}(t)$, we obtain

$$\sum_n c_n \cdot e^{-2\pi inf\Delta t} = \Delta t \sum_j \sum_k r_j s_k e^{-2\pi i(j+k)f\Delta t} = \sum_n \left(\Delta t \sum_{j+k=n} r_j s_k \right) e^{-2\pi inf\Delta t}, \tag{3.10}$$

where we reorganize the double sum over the all pairs j, k as a sum over all diagonals $j + k =$ constant n. Recognizing the complex exponentials as orthogonal functions, we equate the coefficients indexed by n to find

$$c_n = \Delta t \sum_{j+k=n} r_j s_k. \tag{3.11}$$

A simple change of variables $j = n - k$ gives the discrete convolution formula

$$c_n = \Delta t \sum_k r_{n-k} s_k, \tag{3.12}$$

which shows how to compute the samples c_n of the signal $c(t) = (r \bullet s)(t)$ directly from the samples r_j, s_k. We make this formula our definition for the discrete convolution of two sets of signal samples $[r_k], [s_k]$, which we can write as $[c] = [a] \bullet [b]$.

A similar calculation shows that the zero-lag crosscorrelation of two band-limited signals $r(t), s(t)$ is computed from the samples as

$$\int_{-\infty}^{\infty} r(t)s(t)^*\, dt = \Delta t \sum_k r_k s_k^* \tag{3.13}$$

and the energy is given as

$$\int_{-\infty}^{\infty} |s(t)|^2\, dt = \Delta t \sum_k |s_k|^2. \tag{3.14}$$

The *unnormalized* crosscorrelation is defined as

$$cc_j = (r \otimes s)_j = \Delta t \sum_k r_k s_{k+j} \tag{3.15}$$

and, as with the results above, it agrees exactly with the sampled version of the continuous-time crosscorrelation of $(r \otimes s)(t)$. The first notation, cc_j, will be used when the identity of the signals being correlated is either clear or irrelevant, otherwise $(r \otimes s)_j$ will be used.

For discrete crosscorrelation, a normalized measure is often preferred, where the normalization is simply a scale factor that scales the result of Eq. (3.15) into the range $[-1, 1]$, where 1 indicates that the two signals are identical and -1 indicates that they are identical except for a polarity flip (i.e., a scale factor of -1 or a phase rotation of 180°). The normalized discrete crosscorrelation is

$$cc_j = (r \otimes s)_j = \frac{1}{\sqrt{E_r E_s}} \sum_k r_k s_{k+j}, \tag{3.16}$$

where $E_r = \sum_k r_k^2$ is the energy of $[r]$ and, similarly, E_s is the power of $[s]$. It is left as an exercise to confirm that these crosscorrelation values will fall into the range $[-1, 1]$.

Since our original signal had finite energy, the corresponding samples s_k form a sum of squares

$$\sum_{k=-\infty}^{\infty} |s_k|^2 < \infty \tag{3.17}$$

that is finite. The set of all such discrete signals is a complete vector space, known as the Hilbert space $l^2(\mathbf{Z})$, the linear space of square-summable sequences indexed by the integers $\mathbf{Z}$. The vector space has both inner products and convolution defined on it by the above formulas.

Note that, in many applications, the factor Δt is removed for convenience. It is worth emphasizing again that these discrete formulas represent *exact calculations* for Fourier transforms, convolution, inner products, and energies of continuous-time, band-limited signals.

3.1.3 Fourier Transforms and Inverses for Sampled Signals

In this section, we emphasize that Eq. (3.6) is an exact representation of the Fourier transform of a band-limited, continuous-time signal, so we can define the Fourier transform of a sequence of samples s_k as

$$\hat{s}(f) = \Delta t \sum_k s_k \cdot e^{-2\pi i k f \Delta t} \qquad \text{for } -\frac{1}{2\,\Delta t} < f < \frac{1}{2\,\Delta t}. \tag{3.18}$$

Consequently, the Fourier transform of a sequence is just a function on a finite interval in the frequency domain, and happens to agree with the full Fourier transform of a corresponding band-limited, sampled signal in continuous time.

We can recover the samples s_k directly from this function in frequency space, as an integral over that same interval:

$$s_k = \int_{-1/(2\,\Delta t)}^{1/(2\,\Delta t)} \hat{s}(f) e^{2\pi i k f \Delta t}\, df. \tag{3.19}$$

We can see this immediately from Eq. (2.16), using the inverse Fourier transform for continuous time, evaluated at the point $t = k\,\Delta t$. Alternatively, one can observe that the complex exponential functions $e_k(f) = \exp(2\pi i k f \Delta t)$ are orthogonal on the given interval, $[-1/(2\,\Delta t), 1/(2\,\Delta t)]$, for integer values of k.

For sampled signals, these last two formulas are often normalized by setting $\Delta t = 1$.

3.1.4 The Sampling Theorem via the Sampling Comb

For a fixed sampling interval Δt, the *sampling comb*, or just comb function,[5] is defined as an infinite sum of regularly spaced Dirac delta functions

$$\text{Ш}(t) = \sum_{k=-\infty}^{\infty} \delta(t - k\,\Delta t). \tag{3.20}$$

The process of sampling at regular intervals of Δt can be idealized as a product

$$s_{\Delta t}(t) = s(t)\text{Ш}(t), \tag{3.21}$$

where the sampled function $s_{\Delta t}(t)$ is not quite yet the samples but is still a function of the continuous variable t that is nonzero only at the sample locations. To obtain a particular sample, we must perform the integration

$$s_k = \int_{t_k-\epsilon}^{t_k+\epsilon} s_{\Delta t}(t)\, dt = s(t_k), \tag{3.22}$$

where $t_k = k\,\Delta t$ and $\epsilon > 0$ is any positive number less than Δt. Equation (3.22) follows from the sifting property of the delta function as given by Eq. (2.26). In fact, since the only values of $s(t)$ that survive the sampling are the s_k, we can rewrite Eq. (3.21) as

$$s_{\Delta t}(t) = \sum_{k=-\infty}^{\infty} s_k \delta(t - k\,\Delta t). \tag{3.23}$$

While $s_{\Delta t}$ is not quite our sequence of discrete samples, it has exactly the same information content as $[s] = [\ldots, s_{-2}, s_{-1}, s_0, s_1 s_2, \ldots]$ but is still in a form which can be examined with the continuous Fourier transform of Eq. (2.15). In fact, by the convolution theorem, the Fourier transform of $s_{\Delta t}$ must be

$$\hat{s}_{\Delta t}(f) = \left(\hat{s} \bullet \widehat{\text{Ш}}\right)(f)\,; \tag{3.24}$$

[5] Also called the *shah* function (Bracewell, 2000).

that is, the action of time-domain sampling in the frequency domain is to convolve the spectrum of s with the spectrum of c. Calculation of the Fourier transform $\widehat{\text{III}}$ can be found elsewhere (e.g., Papoulis (1984), p. 69); however, it is well known to be

$$\widehat{\text{III}}(f) = f_s \sum_{k=-\infty}^{\infty} \delta(f - kf_s), \tag{3.25}$$

where $f_s = 1/\Delta t$ is called the *sampling frequency*. Thus the sampling comb transforms to a comb in the frequency domain, but the spacing between the "teeth" (the delta functions) is Δt in the time domain and $1/\Delta t$ in the frequency domain. While the mathematical proof of Eq. (3.25) is very challenging, it is not that difficult to demonstrate it numerically. The idea is to compute at a sample size that is much smaller than any of the common sample sizes and let that be a proxy for a continuous signal. In Code Snippet 3.1.1, computation is done at a sample size $\Delta t = 0.0001$ s, which is five times smaller than the smallest standard sample size of 0.0005 s. Three sampling combs are generated, at spacings of 0.05, 0.01, and 0.004 s, where the last one is one of the standard sample sizes. The Fourier transforms of these combs are also computed and plotted, with the result shown in Figure 3.2. Theoretically, the time-domain combs should be continuously sampled and of infinite length, but here they are sampled at 0.0001 s and are 50 s long. Both of these values are extreme compared with typical seismic sample sizes and trace lengths. As can be seen in Figure 3.2, a comb does indeed transform into a comb and the tooth spacings in the time and frequency domains are the inverses of each other. While the time-domain comb has equal-height teeth, this is not quite so in the frequency domain. The slight amplitude modulation of the frequency-domain teeth is caused by the finite length of the time-domain comb. The theoretical comb is infinitely long, and the finite-length comb used here is the product of a finite-length boxcar with the theoretical comb. In the frequency domain, this boxcar truncation is a convolution with a sinc function, which induces the modulation.

Equation (3.24) expresses the frequency-domain consequences of time-domain sampling. Convolution of the spectrum of the continuous signal with $\widehat{\text{III}}(f)$ causes the continuous spectrum to be replicated at every tooth on $\widehat{\text{III}}(f)$. These replicated spectra are centered at the frequencies $\ldots, -2f_s, -f_s, 0, f_s, 2f_s, \ldots$ and there are infinitely many of them. The *sampling theorem* says that if $\hat{s}(f)$ vanishes outside the frequency band $[-f_s/2 \rightarrow f_s/2]$ (recall that $f_{nyq} = f_s/2$), then $s(t)$ can be recovered exactly from $s_{\Delta t}(t)$. This works because the band limit condition means that the spectral aliases are fully separated from the original spectrum.

Code Snippet 3.1.2 performs a numerical demonstration of sampling, and the results are shown in Figures 3.3a and 3.3b. As with Code Snippet 3.1.1, a very small sample size, $\Delta t = 0.0001$ s ($f_{nyq} = 5000$ Hz), is used to create a one-second-long seismic trace that is a proxy for a continuous signal. This "pseudo-continuous" trace is created by the convolution of a random reflectivity with an Ormsby wavelet. The pseudo-continuous trace is then resampled at $\Delta t = 0.004$ s, which has a Nyquist frequency of $f_{nyq} = 125$ Hz. To ensure that the resampling happens without aliasing, the Ormsby wavelet is chosen to have the four characteristic frequencies 5–10–100–120 (Hz), which guarantees that the maximum frequency in the seismic trace is less than f_{nyq}. Figure 3.3a shows the sampling

Code Snippet 3.1.1 Working at a very small sample size (line 2), three different sampling combs are simulated, with tooth spacings (sample sizes) specified on line 3. Theoretically the combs should be infinitely long, and their length is set on line 3 at 50 s. The combs are constructed on line 12 by calling *comb*. Each comb is sampled at `dt` and has teeth spaced at `delt(k)`. For each comb, the Fourier transforms are computed (line 13) using *fft*. The frequency axis for the frequency combs is computed on line 15. The results are shown in Figure 3.2 but the plotting code is not shown.

```
1   %show that a comb tranforms to a comb
2   dt=.0001;%sample rate in time
3   delt=[.05 .01 .004];%time-domain comb spacings
4   tmax=50;%maximum time (length of combs)
5   nt=round(tmax/dt);%initial number of samples
6   nt=2^(nextpow2(nt));%adjust to a power of 2
7   t=dt*(0:nt-1)';%time axis
8   c=zeros(length(t),length(delt));%array for time-domain combs
9   C=c;%array for frequency-domain combs
10  for k=1:length(delt)
11      nt2=round(delt(k)/dt);%number of samples between teeth (time)
12      c(:,k)=comb(t,nt2);%make time comb
13      C(:,k)=abs(fftshift(fft(c(:,k))))/nt;%fft and normalize
14  end
15  f=freqfft(t,nt);%frequency coordinate for the spectrum
```

End Code

signalcode / sampling_comb2.m

process in the time domain. In the upper panel, the pseudo-continuous signal $s(t)$ is shown together with the sampling locations for $\Delta t = 0.004$ s, and in the lower panel is the sampled trace $s_{\Delta t}(t) = s(t)c(t)$. As discussed in conjunction with Eq. (3.21), $s(t)$ and $s_{\Delta t}(t)$ have the same number of samples (10 001),[6] but $s_{\Delta t}(t)$ has only 251 nonzero samples. This means that there are 39 zeros between every two nonzero samples of $s_{\Delta t}(t)$. The Fourier transforms of these two signals are shown in Figure 3.3b as calculated with *fft* in Code Snippet 3.1.2. The *fast Fourier transform*, or FFT, is simply a very efficient implementation of the *discrete Fourier transform*, which is yet to be discussed. As will be seen, the DFT always computes frequencies in the range $[-f_{\text{nyq}} \ldots f_{\text{nyq}}]$, so the effect of the extra zeros in s_{Dt} is to expand the frequency range from $[-125 \ldots 125]$ Hz to $[-5000 \ldots 5000]$ Hz, allowing us to see the spectral aliases created by sampling. In Figure 3.3b, only a very small portion of the computed frequency range is shown, so that only the *principal band* and its first aliases on either side are shown. The occurrence of these spectral aliases is described by Eq. (3.24) because convolution of the band-limited spectrum of the continuous signal $\hat{s}(f)$ with $\widehat{\text{III}}(f)$ causes the former to be replicated at every spike in the latter. Band limiting $\hat{s}(f)$ such that $|f_{\max}| < f_{\text{nyq}}$ means that there will be no overlap between the spectral aliases. In this case we say that the spectrum of the sampled

[6] For a trace length of T and a sample size of Δt, the number of samples is $n = floor(T/\Delta t) + 1$.

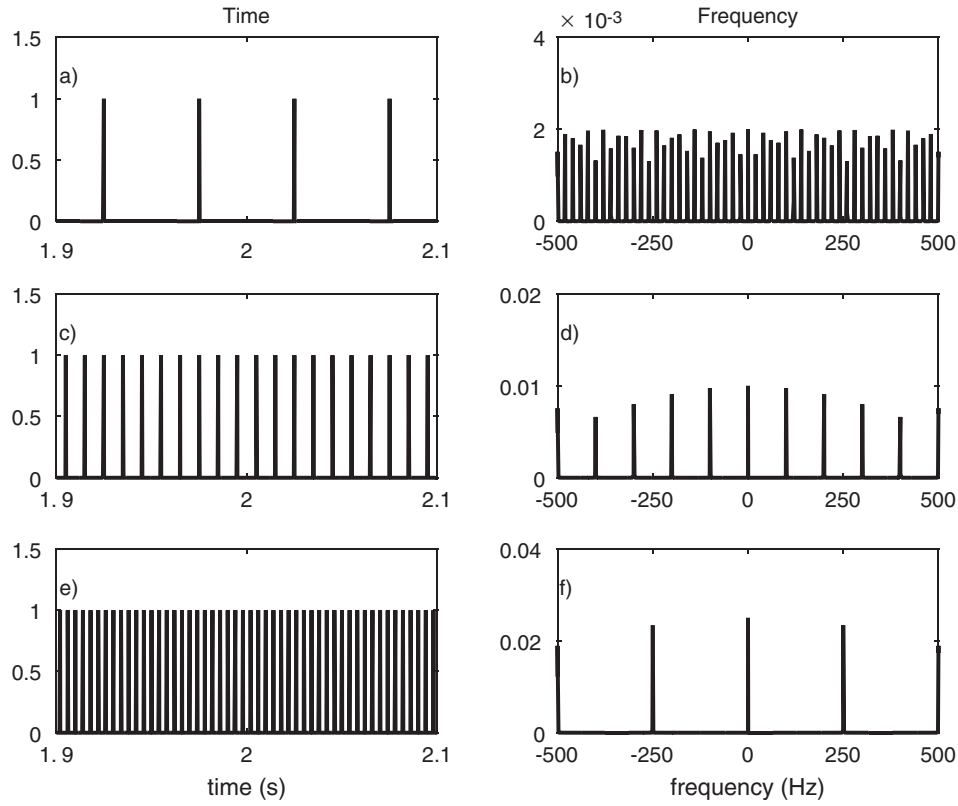


Figure 3.2 Three different sampling combs are shown in the time domain (left column) and the frequency domain (right column). This is the result of the computations in Code Snippet 3.1.1. For $\Delta t = 0.05$ s, the time-domain comb is shown in (a) and its Fourier transform in (b). For $\Delta t = 0.01$ s, the time-domain comb is shown in (c) and its transform in (d). Finally, for $\Delta t = 0.004$ s, the time-domain comb is shown in (e) and its transform in (f). In each case only a small range of the time and frequency spans of the computed signals is shown, to allow visual inspection of the details. In each case also, the time-domain comb has teeth spaced at Δt and its transform has teeth spaced at $1/\Delta t$. The slight variation in height of the teeth of the frequency-domain comb is a modulation caused by the fact that the time-domain comb has finite length.

signal is *not aliased*. We can also speak of the *Nyquist sampling criterion* as requiring two or more samples per period (where period = 1/frequency).

A final statement from the sampling theorem concerns the reconstruction of the continuous signal from its samples. If there is truly no loss of information in the sampling, then this must be possible. The essential idea can be deduced from comparing the upper and lower panels of Figure 3.3b. We seek a frequency-domain method to convert the lower panel into the upper panel, and it should seem obvious: we need to only eliminate the spectral aliases. This can be done by

$$\hat{s}(f) = \hat{s}_{\Delta t}(f)\chi_{\Delta t}(f), \tag{3.26}$$

Code Snippet 3.1.2 Here we compute a synthetic seismogram at a very small time sample size $\Delta t = .0001$ s (to simulate a pseudo-continuous signal) and then sample it at $\Delta t = 0.004$ s. The seismogram uses a 5–10–100–120 Ormsby wavelet that ensures that the seismogram is band limited to less than $f_{\text{nyq}} = 0.5/0.004 = 125$ Hz so that the resampling to $\Delta t = 0.004$ s occurs without aliasing. The pseudo-continuous signal, `s1`, is created on line 7 and the sampling to $\Delta t = 0.004$ s, creating `s2`, happens on line 12. Prior to the extraction of the samples, `s2` is initialized to an array of zeros the same size as `s1`. Thus, between every pair of samples at $\Delta t = 0.004$ s intervals there are $0.004/0.001 - 1 = 39$ zeros. The amplitude spectra of `s1` and `s2` are calculated on lines 14 and 15, and line 18 establishes the indices of the original spectrum. The results are shown in Figures 3.3a and 3.3b but the plotting code is not shown.

```
1   dt1=.0001;%pseudo continuous sample size
2   dt2=.004;%simulated sample size
3   tlen=.4;%length of Ormsby wavelet
4   tmax=1;%length of signal
5   [w,tw]=ormsby(5,10,100,120,tlen,dt1);%bandlimiting Ormsby wavelet
6   [r,t1]=reflec(tmax,dt1,.1,3,4);%reflectivity
7   s1=convz(r,w);%pseudo continuous seismogram
8   s1=s1/max(s1);%normalize
9   %Now sampling
10  s2=zeros(size(t1));%initialize for sampled signal
11  inc=round(dt2/dt1);%indices of desired samples
12  s2(1:inc:end)=s1(1:inc:end);%the actual sampling
13  %Now Fourier transform
14  S1=abs(fftshift(fft(s1)));%amplitude spectrum of continuous signal
15  S2=abs(fftshift(fft(s2)));%amplitude spectrum of sampled signal
16  f1=freqfft(t1,length(t1));%frequency coordinate for spectra
17  fnyq=.5/dt2;%Nyquist frequency
18  ind=near(f1,-fnyq,fnyq);%pointer to the principal frequency band
```

End Code

signalcode / sampling_demo.m

where

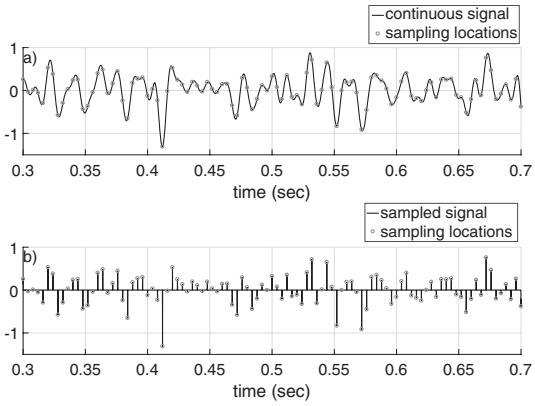
$$\chi_{\Delta t}(f) = \begin{cases} 1, & |f| \le f_{\text{nyq}}, \\ 0, & \text{otherwise} \end{cases} \tag{3.27}$$

is a rescaled boxcar that is unity over the principal band and zero otherwise. The inverse Fourier transform of this boxcar is

$$\check{\chi}_{\Delta t}(t) = \int_{-\infty}^{\infty} \chi_{\Delta t}(f) e^{2\pi i f t}\, df = 2\int_0^{f_{\text{nyq}}} \cos(2\pi f t)\, df = \frac{\sin(\pi t/\Delta t)}{\pi t} = \frac{1}{\Delta t}\text{sinc}\left(\frac{t}{\Delta t}\right), \tag{3.28}$$

and so, using the convolution theorem, we can express Eq. (3.26) in the time domain as

$$s(t) = \left(s_{\Delta t} \bullet \check{\chi}_{\Delta t}\right)(t) = \int_{-\infty}^{\infty} s_{\Delta t}(\tau)\check{\chi}_{\Delta t}(t-\tau)\, d\tau. \tag{3.29}$$



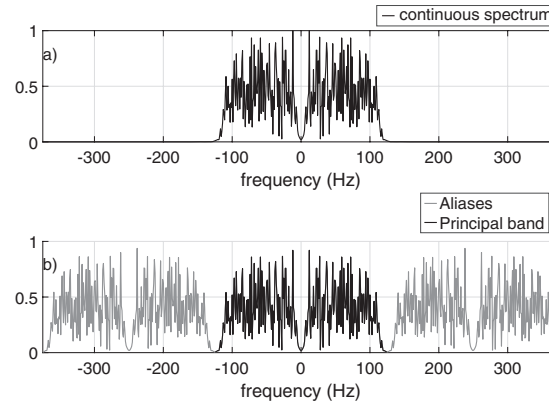


Figure 3.3a (left) The time-domain results for Code Snippet 3.1.2. (a) The original pseudo-continuous signal, $s(t)$, and the locations of the samples at $\Delta t = 0.004$ s intervals. (b) The resulting sampled signal $s_{\Delta t}(t)$, as described by Eq. (3.21). There are 39 zeros between every pair of samples. Only a small portion of the 1 s signal length is shown.

Figure 3.3b (right) The frequency-domain consequences of the time-domain sampling depicted in Figure 3.3a. (a) The spectrum of the original pseudo-continuous signal. (b) The spectrum of $s_{\Delta t}(t)$ as described by Eq. (3.24). Sampling causes infinitely many aliases of the original spectrum, and the first alias on either side of the original *principal band* is shown. The original spectrum was band limited to within $[-125 \rightarrow +125]$ Hz, so we say the sampled signal is *not aliased*. Only a small portion of the total frequency range is shown.

Using Eq. (3.23), this becomes

$$s(t) = \int_{-\infty}^{\infty} \left[\Delta t \sum_{k=-\infty}^{\infty} s_k \delta(\tau - k\,\Delta t) \right] \check{\chi}_{\Delta t}(t - \tau)\, d\tau. \tag{3.30}$$

Interchanging the summation and integration and using Eq. (2.26) to collapse the delta functions gives

$$s(t) = \sum_{k=-\infty}^{\infty} s_k \cdot \mathrm{sinc}\left(\frac{t - k\,\Delta t}{\Delta t} \right). \tag{3.31}$$

Equation (3.31) is exactly Eq. (3.2), which was the smooth *sinc function interpolation* of the sampled signals and in principle will perfectly recover the continuous signal from its samples. Implementation of sinc function interpolation is complicated because $\check{\chi}_{\Delta t}(t)$ is infinitely long and decays only as t^{-1}. However, it is well worth the trouble because any other form of interpolation will cause a loss of accuracy that manifests as inaccurate estimation of the higher frequencies. There are two common approaches: (1) design a finite-length approximate sinc interpolator, and (2) perform the interpolation in the frequency domain. Both possibilities are used frequently and are generally superior to any other form of interpolation, especially linear. Signal interpolation is required in many different circumstances, including plotting, static shifting, normal moveout removal, and migration.

3.1.5 Summary of Time-Domain Sampling Effects

The conclusions reached in this section are of far-reaching importance and deserve a concise summary:

- A signal is said to be *band limited* if its spectrum vanishes outside a finite interval. That is, if $\hat{s}(f) = 0, |f| > f_{\max} > 0$.
- Band-limited signals can be sampled in the time domain without any loss of information. The sample interval, Δt, must satisfy $\Delta t < 1/(2f_{\max})$. Corresponding to any choice of Δt, the *Nyquist frequency*, $f_{\text{nyq}} = 1/(2\,\Delta t)$, is the maximum frequency that can be sampled without aliasing. The Nyquist sampling criterion is that each frequency must have more than two samples per period.
- Time-domain sampling induces periodicity in the frequency domain. That is, the spectrum is replicated at intervals of the sampling frequency $f_s = 1/\Delta t$. The spectral copies are called aliases. If the original spectrum does not overlap with any of its aliases, then we say that the sampled signal is not aliased.
- The continuous signal can be reconstructed from its samples by destroying the frequency-domain aliases. This operation is a frequency-domain multiplication by a boxcar, which, in the time domain, becomes a convolution with a sinc function. This is called *sinc function interpolation* and is the best way to reconstruct a continuous signal from its samples.

3.2 Interpolation, Aliasing, and Resampling

The sampling theorem assures us that a signal can be reconstructed with very high fidelity from its samples, but this is only true if the conditions of the theorem have been met. That is, the signal must be band limited, the sample interval must be chosen properly with respect to the maximum signal frequency, and the continuous signal must be interpolated properly from the samples. This section examines these issues in more detail.

3.2.1 Sinc Function Interpolation

In Section 3.1.1, the theoretical basis for the sampling theorem was outlined and the formula for reconstruction of the band limited, continuous signal from its samples was deduced. That formula is

$$s(t) = \sum_{k=-\infty}^{\infty} s_k \,\text{sinc}\left(\frac{t - k\,\Delta t}{\Delta t}\right) = \sum_{k=-\infty}^{\infty} s_k \frac{\sin(\pi(t - k\,\Delta t)/\Delta t)}{\pi(\Delta t - k\,\Delta t)/\Delta t}, \tag{3.32}$$

where $s(t)$ is the signal being reconstructed and s_k are the samples placed at integer multiples of Δt. The infinite nature of the sum in Eq. (3.32) makes its direct implementation impractical, even when the number of samples is finite, because there still may be many, many samples. It is desirable to have a more local method that uses only samples near time

t to compute $s(t)$. The sinc function has its maximum when its argument is zero, which is when $k = t/\Delta t$. Since k counts samples, it can only assume integer values and t is presumably between two samples, so then this maximum occurs between samples. Let k_t be the largest integer for which $k_t\,\Delta t \le t$; then the sinc function argument can be rewritten as

$$\frac{t - k\,\Delta t}{\Delta t} = \frac{t - (k_t + k - k_t)\,\Delta t}{\Delta t} = \frac{t - (k_t + j)\,\Delta t}{\Delta t}, \tag{3.33}$$

where $j = k - k_t$ counts samples relative to k_t. Then a reasonable modification to Eq. (3.32) is

$$s(t) \approx \sum_{j=-n+1}^{n} s_j \operatorname{sinc}\left(\frac{t - (k_t + j)\,\Delta t}{\Delta t}\right) g_\mu\left(\frac{t - (k_t + j)\,\Delta t}{\Delta t}\right), \tag{3.34}$$

where

$$g_\mu(t) = e^{-\mu^2 t^2/n^2} \tag{3.35}$$

is a Gaussian window with standard deviation $\sigma = n/\mu$, with $\mu = 2$ being a good choice. Equation (3.34) approximates the infinite sum in Eq. (3.32) with a finite sum of $N = 2n$ points evenly distributed on both sides of the interpolation site. Rather than simply truncate the sinc function at roughly n points away from its maximum, a Gaussian window is used to taper the sinc smoothly toward zero at the truncation point. The amount of taper is controlled by μ, with $\mu = 2$ meaning that the Gaussian is two standard deviations away from maximum at the truncation point. At two standard deviations, a Gaussian has the value $e^{-4} \approx 0.0183$.

The application of Eq. (3.34), for the case of $N = 16$ and $\Delta t = 0.004$ s, is illustrated in Figure 3.4a. Here the interpolation site is at $t = 0.743$ s, which is three-quarters of the way from the sample at 0.740 s to the sample at 0.744 s. Sample k_t is the sample at 0.740 s, and the 16 samples used in the interpolation are shown with dark black circles (gray circles are unused samples). There are $n = 8$ samples used on each side of the interpolation site. The sinc function and the Gaussian window are shown positioned with their maxima at the interpolation site. The interpolated value is the sum of the N indicated samples multiplied by their weights, which are the values of the sinc function at the samples' positions. Shown are both the truncated sinc and the tapered sinc, with interpolation values for both. In this case the two results are very similar and both fall near the original continuous signal.

This interpolation with a Gaussian-windowed sinc function is implemented in *interpbl*. For efficiency, this function builds a "sinc table" of sinc function values sampled at $\Delta t/50$, where Δt is the sample size of the input samples. Then sinc function weights are chosen from this table as needed in a nearest-neighbor lookup process. This avoids excessive recomputation of sinc function values when many interpolated samples are needed. MATLAB offers a number of interpolation methods, such as linear, spline, and cubic, through its *interp1* function but none of these are specifically for band-limited signals. *interpbl* augments the standard methods and is usually preferable for seismic data. The syntax for *interpbl* is similar to that for *interp1* and is

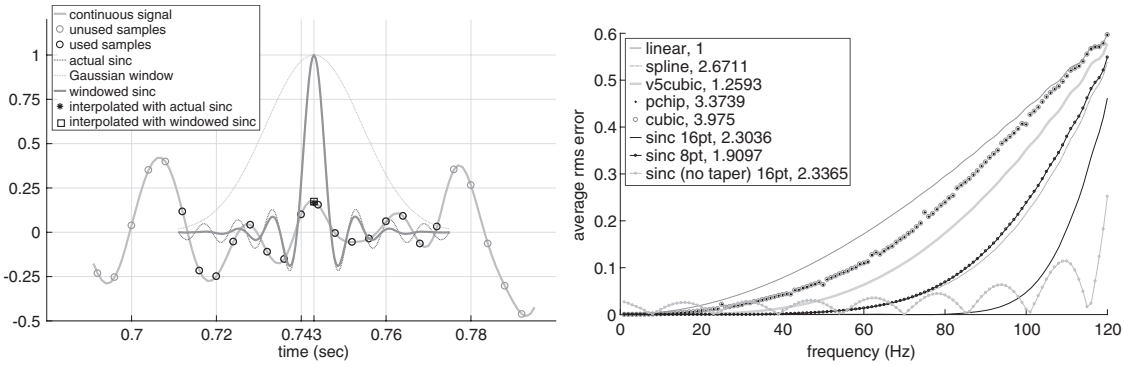


Figure 3.4a (left) The process of sinc function interpolation, as defined by Eq. (3.32), is illustrated for the interpolation site at $t = 0.743$ s. The underlying continuous, band-limited signal has samples taken every 0.004 s as denoted by circles. The darker circles show the 16 samples actually used in this interpolation. The windowed sinc function provides weights for these samples, and the peak of the sinc function is placed at the time of the interpolation.

Figure 3.4b (right) The various interpolation methods contained in MATLAB's *interp1* function are compared with *interpbl*, which does sinc function interpolation. For each integer frequency between 1 and 120 Hz, synthetic sine waves were generated and sampled at $\Delta t = 0.004$ s and then interpolated to $\Delta t = 0.002$ s. Shown are the average rms (root mean square) errors for each method versus frequency. For the sinc function interpolation, results are shown for 16-point and 8-point calculations, with the 16-point result shown with and without Gaussian tapering of the sinc function. See Figure 3.4a to understand the process.

usually `sint=interpbl(t,s,tint,n)`, where `t` and `s` are the times and values of the sampled signal, `tint` is a vector of times at which interpolated values are desired, and `sint` is the same-sized vector of interpolated values. The final input, `n`, is the n used in Eq. (3.34). For example, when $n = 4$, the process is referred to as 8-point sinc function interpolation.

Figure 3.4b compares *interpbl* with all of the methods found in *interp1*. (The various MATLAB methods will not be explained here.) This test operated at $\Delta t = 0.004$ s and generated mathematical sine waves for all integer frequencies from 1 to 120 Hz. Each interpolation method was then used to compute interpolated samples for $\Delta t = 0.002$ s. Since the sine waves are mathematically defined, the errors of each interpolation method can be easily computed. In the legend of Figure 3.4b, following the name of each interpolation method is a number giving the relative computation time for the method. Linear interpolation is the fastest but also the least accurate. The other methods are all roughly comparable in speed but not in accuracy. By far the most accurate is the 16-point sinc function interpolation, while an 8-point sinc function is similar in performance to MATLAB's spline interpolator. The various other built-in interpolators fall well short of even the 8-point sinc in accuracy. Also shown is the result of 16-point sinc function interpolation when the sinc function is not tapered with the Gaussian window. Clearly this tapering is a good thing. None of the interpolation methods performs all the way to f_{nyq}. While sinc

function interpolation is usually superior, it is necessary to use a sufficiently long operator and have a robust algorithm. Simple truncation rather than tapering of the sinc function gives poor results.

The difficulty of accurate interpolation at high frequencies is a major contributor to high-frequency signal loss or distortion in data processing. Any particular trace might undergo interpolation three or more times before it contributes to a final image. Static shifts, normal moveout removal, and imaging algorithms may all require trace interpolation. For this reason, many seismic processors prefer to oversample their data by a factor of two or more. That is, if $f_{\max}$ is the maximum signal frequency, then a conservative approach is to choose Δt such that $f_{\max} < \frac{1}{2} f_{\text{nyq}}$ rather than $f_{\max} < f_{\text{nyq}}$. As a rule of thumb, most data-processing algorithms require more samples than the Nyquist limit of two per period or wavelength. The oversampling approach offers a measure of insurance against this loss of performance.

3.2.2 Frequency-Domain Aliasing

When the seismic source has a known band limit, as is the case with a Vibroseis source, it seems a simple matter to choose the sample interval using the Nyquist criterion according to $f_{\max} \leq c f_{\text{nyq}} = c/(2\,\Delta t)$, where $c = 1$ in theory but $c = \frac{1}{2}$ is a more conservative choice. However, in reality, a seismic vibrator is an imperfect machine and always generates frequencies, called harmonics, that can be two or three times higher than $f_{\max}$. With other seismic sources such as dynamite and weight drops, $f_{\max}$ is entirely unknown. For these reasons, temporal sampling is generally done after passing the continuous signal through an *antialias filter* designed to reject frequencies higher than f_{nyq} before sampling. A typical antialias filter has a passband of $|f| < 0.6 f_{\text{nyq}}$ and a steep rolloff (attenuation curve) for $|f| > 0.6 f_{\text{nyq}}$ such that it is at least 60 dB down by f_{nyq}. This is yet another reason to choose the constant $c \leq \frac{1}{2}$.

With the use of antialias filters, temporal sampling can almost always be done without aliasing. Still, it is useful to analyze what an aliased signal looks like because we will find later that spatial sampling is almost always associated with aliasing. Code Snippet 3.2.1 details an aliasing experiment, with the results shown in Figures 3.5a and 3.5b. A 200 Hz sine wave is initially sampled at $\Delta t = 0.001$ s and then downsampled to $\Delta t = 0.002$ s by selecting every other sample, and again downsampled to $\Delta t = 0.004$ s by selecting every fourth sample. For $\Delta t = 0.001$, $f_{\text{nyq}} = 500$ Hz, while for $\Delta t = 0.002$ s, $f_{\text{nyq}} = 250$ Hz, and for $\Delta t = 0.004$ s, $f_{\text{nyq}} = 125$ Hz. Therefore we expect no aliasing for the first downsampling but the second downsampling should have definite aliasing. For $f_{\text{nyq}} = 125$, 200 Hz is 75 Hz above Nyquist and it should alias to 75 Hz below. Figure 3.5a shows that it does indeed alias to 50 Hz. In the time domain, the 50 Hz sine wave can be seen directly as the curve that connects every fourth point. In the frequency domain, there is a clear spike at 50 Hz as expected. Figure 3.6 demonstrates that sinc function interpolation cannot undo the aliasing even though it does a very fine job of smoothing out the unaliased signals.

As a second experiment, consider the sampling at $\Delta t = 0.004$ s of five sine waves with frequencies of 105, 115, 125, 130, and 140 Hz. From the previous results, and also from

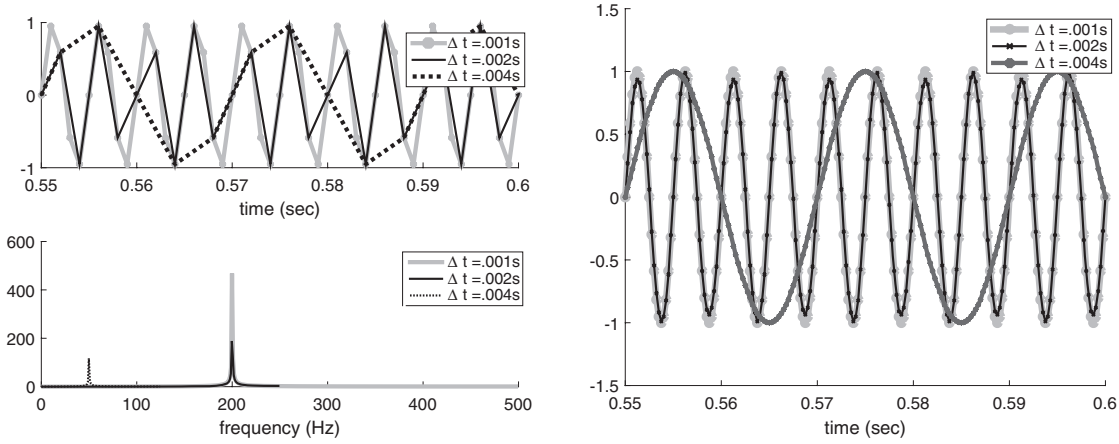


Figure 3.5a (left) A 200 Hz sine wave is shown with samples at three different Δt's: 0.001 s, 0.002 s, and 0.004 s. The corresponding Nyquist frequencies are 500 Hz, 250 Hz, and 125 Hz. Therefore aliasing is expected at $\Delta t = 0.004$ s. Top: The time-domain picture. Sample locations are shown for $\Delta t = 0.001$ s. Notice how the coarser sample rates simply skip over samples. Bottom: The frequency-domain picture. The 200 Hz sine wave sampled at $\Delta t = 0.004$ s causes an aliased spike at 50 Hz.

Figure 3.5b (right)The three sine waves of Figure 3.5a are shown after sinc function interpolation to $\Delta t = 0.00025$ s. Interpolation recovers the 200 Hz sine wave to great accuracy except for the aliased case.

the perspective of sampling theory, a frequency $f = f_{\text{nyq}} + \delta f$, where $\delta f > 0$, should produce an alias in the amplitude spectrum at $f_{\text{a}} = f_{\text{nyq}} - \delta f$. (Note that the actual alias at a positive frequency comes from the negative portion of the first alias to the right (Figure 3.3b). This affects only the phase spectrum.) Thus 130 Hz should produce an aliased peak in the amplitude spectrum at 120 Hz, and 140 Hz should have an alias at 110 Hz. Figure 3.6 shows the results of this experiment (the code is left as an exercise), and it is clear that the two aliased frequencies have resulted in spectral peaks that are in between the peaks of the unaliased signals. This was by design and clearly illustrates that $f_{\text{nyq}} + \delta f$ aliases to $f_{\text{nyq}} - \delta f$.

Exercises

3.2.1 Write a code to produce a figure similar to Figure 3.6, making sure to sample sine waves (see Code Snippet 3.2.1 for help). Run your code for the frequencies 105, 115, 125, 130, and 140 Hz and a second time for 105, 115, 125, 135, and 145 Hz. Why are four spectral peaks produced the first time and only two the second time? Then, change your code to sample cosine waves instead. Why do the cosine waves produce a large peak at f_{nyq} while the sine waves do not?

Code Snippet 3.2.1 A frequency aliasing experiment is conducted by generating a 200 Hz sine wave initially at $\Delta t = 0.001$ s (line 6) and then resampling it to $\Delta t = 0.002$ s (line 8) and $\Delta t = 0.004$ s (line 11). Aliasing is expected in the third case. The one-sided spectra of all three signals are computed on lines 14–16. Finally, the sampled signals are all sent through sinc function interpolation to be resampled to $\Delta t = 0.00025$ s to simulate recovering the continuous signal. Results are shown in Figures 3.5a and and 3.5b.

```
1   %An aliasing experiment
2   %make a sine wave of 200 Hz sampled at 1 mil, 2 mils and 4 mils
3   dt=.001;%base sample rate
4   t1=0:dt:1;%time coordinates at 1 mil
5   f=200;%frequency of sine wave
6   s1=sin(2*pi*f*t1);%base sine wave
7   %sample at 2 mil
8   s2=s1(1:2:end);%take every other sample from base sine wave
9   t2=t1(1:2:end);
10  %sample at 4 mil
11  s4=s1(1:4:end);%take every 4th sample from base sine wave
12  t4=t1(1:4:end);
13  %spectra (one-sided)
14  [S1,f1]=fftrl(s1,t1);
15  [S2,f2]=fftrl(s2,t2);
16  [S4,f4]=fftrl(s4,t4);
17
18  % attempt recovery of the continuous signal by interpolation
19  % sinc function interpolation
20  dt2=dt/4;%we will interpolate to 0.00025s
21  tint=t1(1):dt2:t1(end);%interpolation locations
22  s1i=interpbl(t1,s1,tint);
23  s2i=interpbl(t2,s2,tint);
24  s4i=interpbl(t4,s4,tint);
```

End Code

signalcode / frequency_domain_aliasing1 .m

3.3 Discrete Convolution

Consider the digital sampling of all three of the functions $a(t), b(t), c(t)$ involved in Eq. (2.14) defining the convolution operation. Let Δt be the *temporal sample interval* and let each of $a(t), b(t)$, and $c(t)$ be represented by signals of $2N + 1$ discrete samples $[a_k]$, $[b_k]$, and $[c_k]$ with $k = 0, \pm 1, \ldots, \pm N$ so that $t_k = k\,\Delta t$ ranges from $-N\,\Delta t$ to $+N\,\Delta t$. Then, a *discrete* version of Eq. (2.14) can be written as the approximation

$$c_j \approx \Delta t \sum_{k=-N}^{N} a_k b_{j-k}, \tag{3.36}$$

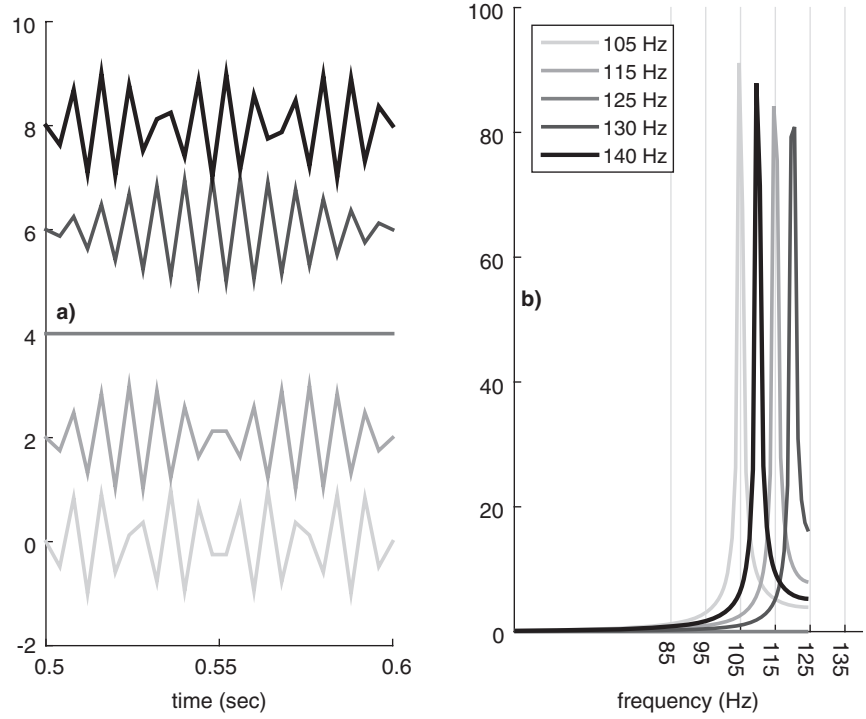


Figure 3.6 (a) Five sine waves of frequencies 105, 115, 125, 130, and 140 Hz are shown in the time domain after sampling at $\Delta t = 0.004$ s. The Nyquist frequency is $f_{\text{nyq}} = 125$ Hz, so the last two are aliased. (b) The one-sided amplitude spectra, computed with $fftrl$, of the sine waves of panel a. The frequency of 130 Hz has produced the alias at 120 Hz, while the 140 Hz signal is aliased to 110 Hz.

where the constant Δt has been taken outside the summation. In most implementations of digital signal theory, Δt is a constant throughout the application and it is customary to write equations like (3.36) with Δt set to unity. The reason this is only an approximation to the continuous convolution is that the sampled signals only approximate the original signals $a(t), b(t), c(t)$. Nevertheless, as noted early, the formula is exact for band-limited signals, so the more common discrete of two sequences $[a_k], [b_k]$ will be defined as the sequence

$$c_j = \sum_{k=-\infty}^{\infty} a_k b_{j-k}, \quad \text{for} \quad j = 0, \pm 1, \pm 2, \pm 3, \ldots, \tag{3.37}$$

where here the range of summation is expanded to include all possible indices for the sequences.

Exercises

3.3.1 Study Example 3.1 and then compute the convolution of $w = [0.1, 0.3, 0.1]$ with $r = [1, -0.3, 0, -0.5, 0.8, -0.8]$. Make a sketch of the process of reversing w and sliding it past r. Redraw this sketch for each output sample.

3.3.2 What are the rules similar to those for continuous convolution in Exercise 2.3.3 for a correct discrete convolution sum?

Example 3.1 A simple model of a seismic trace, s_k, is that it represents a reflectivity series, r_k, convolved with a wavelet, w_k. That is, $s_j = \sum_k r_k w_{j-k}$. Let $r = [1, 0.2, -0.1, -0.5, 0.5, 0, -1]$ and $w = [1, -0.5, 0.2]$ and compute s_j. Assuming that both $[r]$ and $[w]$ begin with sample number 0, s_0 is given by $s_0 = \sum_k r_k w_{0-k}$. The only nonzero contribution to this sum is for $k = 0$, with the result $s_0 = r_0 w_0 = 1$. Below, each element of $[s]$ is worked out. In each case, the limits on the summation are adjusted to include only the terms that involve nonzero contributions from $[w]$. This means that a maximum of three elements occur in each sum. The entire process can be visualized by reversing $[w]$ and sliding it past $[r]$. At each position, the samples that align are multiplied and summed:

$$s_0 = \sum_{k=0}^{0} r_k w_{0-k} = r_0 w_0 = 1,$$

$$s_1 = \sum_{k=0}^{1} r_k w_{1-k} = r_0 w_1 + r_1 w_0 = -0.5 + 0.2 = -0.3,$$

$$s_2 = \sum_{k=0}^{2} r_k w_{2-k} = r_0 w_2 + r_1 w_1 + r_2 w_0 = 0.2 - 0.1 - 0.1 = 0,$$

$$s_3 = \sum_{k=1}^{3} r_k w_{3-k} = r_1 w_2 + r_2 w_1 + r_3 w_0 = 0.04 + 0.05 - 0.5 = -0.41,$$

$$s_4 = \sum_{k=2}^{4} r_k w_{4-k} = r_2 w_2 + r_3 w_1 + r_4 w_0 = -0.02 + 0.25 + 0.5 = 0.73,$$

$$s_5 = \sum_{k=3}^{5} r_k w_{5-k} = r_3 w_2 + r_4 w_1 + r_5 w_0 = -0.1 - 0.25 + 0 = -0.35,$$

$$s_6 = \sum_{k=4}^{6} r_k w_{6-k} = r_4 w_2 + r_5 w_1 + r_6 w_0 = 0.1 + 0 - 1 = -0.9,$$

$$s_7 = \sum_{k=5}^{6} r_k w_{7-k} = r_5 w_2 + r_6 w_1 = 0 + 0.5 = 0.5,$$

$$s_8 = \sum_{k=6}^{6} r_k w_{8-k} = r_6 w_2 = -0.2.$$

3.3.1 Numerical Convolution

Let us examine the application of Eq. (3.37) to some sampled signals. MATLAB provides the *conv* command, which directly implements Eq. (3.37). However, there are two issues that need to be considered when using this command. First, notice that there is no explicit notation for time t in Eq. (3.37), while there is in Eqs. (2.14) and (2.13). As a consequence, *conv* assumes the first sample in each time series which occurs at $t = 0$. This seemingly benign assumption can cause bookkeeping difficulties when dealing with geophysical data. Second, Eq. (3.37) requires that the length of $[c]$ be related to the lengths of $[a]$ and $[b]$ by

$$length([c]) = length([a]) + length([b]) - 1. \tag{3.38}$$

Thus, if $[a]$ has 401 samples and $[b]$ has 1001 samples, then $[c]$ will have 1401 samples. This means that every time a seismic trace is convolved with a filter, the filtered trace must be longer than the original. In a seismic processing system where millions, sometimes billions, of traces are being processed and filtered, it is impractical to allow the signals to grow in length.

Consider again Figure 2.4a and compare the lengths of $[r]$, $[s_m]$, and $[s_z]$ in the lower panel. Since $[s_m]$ and $[s_z]$ have the same length as $[r]$, creation of this figure involved something more than just the application of *conv*. To clarify this, examine Figure 3.7, which recreates Figure 2.4a using only *conv* and plots the signals versus sample number

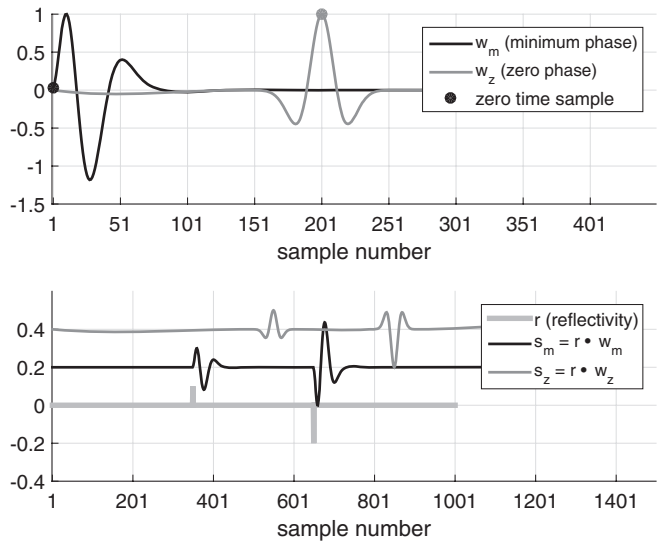


Figure 3.7 A re-creation of Figure 2.4a using only *conv*. The results of `sm=conv(r,wm)` and `sz=conv(r,wz)` are shown plotted versus sample number. The top panel shows the two wavelets versus sample number and the bottom panel shows `r`, `sm`, and `sz` versus sample number. In the top panel, a large dot shows the location of the zero-time sample.

instead of time. In the upper panel we see that, while both wavelets have 401 samples, $[w_m]$ shows the wavelet with significant amplitude at sample number 1 while, in contrast, $[w_z]$ has its main energy in the middle of the signal. In contrast, the upper panel of Figure 2.4a shows the correct physical picture because there the wavelets are aligned properly in time. In the lower panel, we see that $[r]$ has 1001 samples and $[s_m]$ and $[s_z]$ both have 1401 samples as Eq. (3.38) requires. We also see that, when plotted versus sample number, the wavelets in $[s_m]$ align with the corresponding reflection coefficients while those in $[s_z]$ do not. The alignment issue arises because we choose to designate the central sample (number 201) in $[w_z]$ as time zero rather than the first sample.

Both the alignment issue and the length issue can be resolved by intelligently truncating $[s_m]$ and $[s_z]$ to have the same length as $[r]$. More generally, plotting and processing both causal and noncausal signals properly are facilitated in this text and the *NMES Toolbox* by requiring that each signal be described by two vectors, one giving the sample values and the other giving the sample times or time coordinates. Commonly in other texts only the first vector is used, with the second implied to always give time zero at the first sample, but this leads to confusion and a lack of generality. If a gather of traces all have the same time coordinates, then only one time coordinate vector is needed for the entire gather. This is a small price to pay.

The intelligent truncation after convolution is accomplished by the two commands *convm* and *convz*. Both of these are simply wrapper functions that first call *conv* and then truncate the result. For the case of Figure 3.7, both $[s_m]$ and $[s_z]$ need to be shortened by 400 samples, which is the length of the wavelet less one sample. However, for $[s_m]$ the appropriate 400 samples are all at the end of the signal, while for $[s_z]$ it is appropriate to delete 200 samples from the beginning and another 200 from the end. These truncations are accomplished by *convm* for $[s_m]$ and *convz* for $[s_z]$. In using these new commands, the first argument defines the desired length of the result and is expected to be longer than the second input. *convm* shortens the convolved result entirely by deleting samples from the end, as is appropriate if the second input is causal. Alternatively, *convz* by default will delete samples symmetrically from the beginning and the end. This default behavior can be modified by giving a third input, which defines which sample in the second argument occurs at time zero. So, in summary, to produce the results in Figure 2.4a, use `sm=convm(r,wm)` and `sz=convz(r,wz)`, while the results in Figure 3.7 are produced by `sm=conv(r,wm)` and `sz=conv(r,wz)`.

Convolution by Polynomial Multiplication: The Z-Transform

There is a convenient representation of discretely sampled signals that allows convolution to be done through a simple process of multiplication. Suppose that a signal is given by

$$[a] = [a_{-M}, a_{-M+1}, \ldots a_{-1}, a_0, a_1, \ldots a_k, \ldots a_N],$$

where the subscript k denotes the time of the sample by $t_k = k\,\Delta t$, with Δt being a constant called the sample interval. Thus this signal has M samples in the past, N samples in the future, and one sample at $t = 0$. We identify this signal with a (Laurent) polynomial in a

complex variable z by the relation

$$A(z) = a_{-M}z^{-M} + a_{-M+1}z^{-M+1} + \cdots + a_{-1}z^{-1} + a_0 + a_1 z + \ldots a_N z^N = \sum_{k=-M}^{N} a_k z^k. \quad (3.39)$$

The resulting function $A(z)$ is called the *z-transform* of the discrete signal $[a]$. The specification that z be a complex number, having both real and imaginary parts, will become critical when we develop the connection between the z-transform and the discrete Fourier transform. For now, suppose we take $z = e^{-i\omega\,\Delta t}$ (where $\omega = 2\pi f$ is the frequency in radians/second); then $\sum_k a_k z^k = \sum_k a_k e^{-i\omega k\,\Delta t}$. The last sum is a form of the DFT (see Section 3.4). This means that we will attach special significance to the behavior of our polynomials when $z = e^{-i\omega\,\Delta t}$. These z values are all complex numbers of magnitude 1 ($\left|e^{-i\omega\,\Delta t}\right| = 1$) and lie on the *unit circle* in the complex plane. With this definition, the exponent of z tells the time of the corresponding sample in the signal. Thus a causal signal, which vanishes for $t < 0$, has only nonnegative powers of z in its z-transform:

$$A_{\text{causal}}(z) = \sum_{k=0}^{N} a_k z^k. \quad (3.40)$$

Conversely, the presence of any negative powers in the z-transform indicates that the corresponding signal is noncausal. The inverse z-transform amounts to simply reading the polynomial coefficients and reconstructing the signal.

Now, recall the rule for the multiplication of two polynomials; for example, suppose $[a]$ and $[b]$ are both causal signals with $N = 2$, and examine the multiplication of their z-transforms

$$\begin{aligned} A(z)B(z) &= \left(a_0 + a_1 z + a_2 z^2\right)\left(b_0 + b_1 z + b_2 z^2\right) \\ &= a_0 b_0 + a_0 b_1 z + a_0 b_2 z^2 + a_1 z b_0 + a_1 z b_1 z + a_1 z b_2 z^2 + a_2 z^2 b_0 \\ &\quad + a_2 z^2 b_1 z + a_2 z^2 b_2 z^2; \end{aligned} \quad (3.41)$$

then, collecting terms in like powers of z gives

$$\begin{aligned} &A(z)B(z) \\ &= a_0 b_0 + (a_0 b_1 + a_1 b_0)\, z + (a_0 b_2 + a_1 b_1 + a_2 b_0)\, z^2 + (a_1 b_2 + a_2 b_1)\, z^3 + a_2 b_2 z^4. \end{aligned} \quad (3.42)$$

Examination of this result shows that the coefficient of each power of z is composed of all combinations of the coefficients of $A(z)$ and $B(z)$ whose subscripts sum to the value of the exponent of z. In fact, the general pattern for the coefficient of z^j is $\sum_k a_k b_{j-k}$, where the sum extends over all relevant values. (See also Exercise 3.3.2.) Comparison with Eq. (3.37) allows the conclusion that *the product of the z-transforms of two signals gives the z-transform of the convolution of those signals*. Thus we have a new way to accomplish a discrete convolution of two signals $[c] = [a] \bullet [b]$. We form the z-transforms

$A(z)$ and $B(z)$, then calculate the polynomial multiplication $C(z) = A(z)B(z)$, and finally construct the signal $[c]$ by reading off the coefficients of $C(z)$.

These concepts establish a link between the theory of discrete digital filters and the algebra of polynomials. This link has far-reaching implications in signal theory because it allows the many properties of polynomials to be applied to signals. Among the most important of these properties are:

1. All polynomials can be factored and represented as a product of their factors. Signals can thus be factored into a set of simpler signals.
2. The multiplication of polynomials has already been shown to correspond to the convolution of two signals. Since polynomial division is also defined, this implies that a new signal can be created that undoes the convolution with a given signal. This new signal is called an *inverse filter*.

Consider the first point. A quadratic polynomial can be factored by finding its roots using the *quadratic formula*. That is, if $B(z) = b_2z^2 + b_1z + b_0$, then the two roots of $B(z)$ are $z_1 = (-b_1 + \sqrt{b_1^2 - 4b_2b_0})/(2b_2)$ and $z_2 = (-b_1 - \sqrt{b_1^2 - 4b_2b_0})/(2b_2)$. This enables the polynomial to be written in the equivalent factored form $B(z) = b_2(z - z_1)(z - z_2)$.[7] The fundamental theorem of algebra assures us that an Nth-order polynomial will always have N roots, although they may be complex numbers. This means that a signal $[b]$ of length $N + 1$ is completely defined by either its $N + 1$ digital samples $b_0, b_1, \ldots, b_N$ or the N roots of $B(z)$, which are $z_1, z_2, \ldots, z_N$, plus knowledge of any one sample. Thus, if we know the roots of the z-transform of a digital filter, we know almost everything there is to know about the filter. While it is easy to factor a quadratic by hand, it gets increasingly difficult for $N > 2$; however, computers can factor polynomials of order in the hundreds or even thousands. In MATLAB, the `roots` command does this. If the z-transform of a signal can be rendered into a series of elementary factors, then the corresponding signal can be written as the convolution of a series of elementary filters.

Now, the second point, which has far-reaching implications. If we can convolve two signals by multiplying their z-transforms, then we can undo a convolution by an appropriate polynomial division. That is, if $[c] = [a] \bullet [b]$, and we wish to recover $[b]$ from $[c]$ given knowledge of $[a]$, we can do so by polynomial division. That is, $[b]$ will have the z-transform $b(z) = c(z)/a(z)$. Using polynomial division, we can write this as

$$B(z) = \frac{C(z)}{A(z)} = A^{-1}(z)c(z), \tag{3.43}$$

where we have defined the *inverse filter* as the signal corresponding to

$$A^{-1}(z) = \frac{1}{A(z)}. \tag{3.44}$$

In order to turn $A^{-1}(z)$ into a realizable filter, it helps to express it as a power series in z, and this is where polynomial division is used. Such power series representations will generally

[7] The presence of "b_2" here means that we need to know the two roots and one coefficient to unambiguously represent the signal.

have infinitely many terms, that is, they will be infinite series. There are many infinite series that do not converge as the number of terms goes to infinity, so not all signals have realizable inverse filters, because their corresponding power series in z diverges. A signal $[a]$ is called *stable* if it has finite energy, meaning that $\sum_k a_k^2$ is finite (where the sum extends over all samples of the signal). All finite-length signals, and hence all recorded signals, are stable but their inverses might not be. This suggests that we need to study which signals have stable inverses, and so we must examine polynomial division.

In general, polynomial division can be a complicated process, but there is one case where it is very simple and that is when the signal of interest has only two samples. This is the so-called *two-sticker* signal and it is important because, by factorization, we can render the z-transform of any signal into a product of two-stickers. Let the signal of interest be given by $[a] = [a_0, a_1]$, which has the z-transform $a(z) = a_0 + a_1 z$, and we ask whether or not $A^{-1}(z) = 1/(a_0 + a_1 z)$ corresponds to a stable signal. To answer this question, we write $A^{-1}(z) = 1/(a_0 + a_1 z) = (1/a_0)(1/(1 + \alpha z))$, where $\alpha = a_1/a_0$. The factor $1/a_0$ is irrelevant to the question of stability, since any finite scale factor cannot affect the finiteness of the signal's energy. Therefore, without loss of generality, we take $a_0 = 1$ and consider

$$A^{-1}(z) = \frac{1}{1 + \alpha z} = 1 - \alpha z + (\alpha z)^2 - (\alpha z)^3 + \ldots. \tag{3.45}$$

This infinite series is an example of a geometric series and is sometimes called *the* geometric series. The defining characteristic of any geometric series is that the ratio of any two consecutive terms is a constant for the series; in this case the ratio is αz. To show that the series does indeed represent $1/(1 + \alpha z)$, we assume the series converges to a finite value, which we call S. Then consider the algebraic manipulations

$$\begin{aligned} S &= 1 - \alpha z + (\alpha z)^2 - (\alpha z)^3 + \ldots, \\ 1 - S &= \alpha z - (\alpha z)^2 + (\alpha z)^3 + \ldots, \\ \frac{1 - S}{\alpha z} &= 1 - \alpha z + (\alpha z)^2 - (\alpha z)^3 + \cdots = S. \end{aligned}$$

So, $(1 - S) = S\alpha z$, from which it follows that $S = 1/(1 + \alpha z)$. However, these symbolic manipulations only have meaning if the series converges, and the necessary and sufficient condition for this is that $|\alpha z| < 1$. Early in this section, it was mentioned that we are most concerned about the behavior of our filters on the unit circle, so we will take $|z| = 1$ and therefore our stability condition is that $|\alpha| < 1$. This means that our two-sticker must have the second sample smaller than the first, that is, $a_1 < a_0$. So, this analysis shows that the two-sticker $[a] = [a_0, a_1]$ has a stable, causal inverse if, and only if, $|a_0| > |a_1|$. The zero of this elementary polynomial is located at $z_1 = \alpha^{-1} = a_0/a_1$, which is a point outside the unit circle in the complex plane. Now, a longer signal can always be factored and represented as the convolution of all of its two-stickers. The inverse of this signal will then be the convolution of all of the inverses of the two-stickers. This inverse will only be stable if the inverse of each and every two-sticker is stable. Thus, we conclude that an arbitrary signal of finite length has a stable, causal inverse only if all of the zeros of its z-transform lie outside the unit circle. This leads to the following theorem:

Theorem *(Polynomial z-transforms.) A causal, stable signal whose z-transform is a polynomial with zeros outside the unit circle is necessarily minimum phase.*

This is a result of the theorem in Section 3.5 on discrete minimum phase and gives another characterization of an important physical concept. It is tempting to conclude that a diagnostic test for minimum phase is that all of the zeros of the z-transform must lie outside the unit circle. However, a problem arises because the inverse of the polynomial is a rational function, which has no zeros anywhere, so we cannot test whether it too is minimum phase. For instance, the two-sticker with $A(z) = a_0 + a_1 z$ has a zero at $-a_0/a_1$, and the function $A^{-1}(z)$ has a simple pole there, and no zeros. Fortunately, we can expand the zeros result to include signals whose z-transform can be expressed as a rational function

$$A(z) = \frac{A_n(z)}{A_d(z)}, \tag{3.46}$$

where $A_n(z)$ and $A_d(z)$ are polynomials of arbitrary order containing both positive and negative powers of z. Then we characterize this generalized signal by the zeros of both $A_n(z)$ and $A_d(z)$, where the zeros of $A_d(z)$ are called *poles* of $A(z)$. Now we can give another characterization of minimum phase:

Theorem *(Rational z-transforms.) A signal whose z-transform is a rational function with all its zeros and poles outside the unit circle is necessarily minimum phase.*

It is important to note that not all causal, stable signals have z-transforms that can be expressed as rational functions. Indeed, there are interesting signals whose z-transform defines an analytic function on the unit disk that may have infinitely many zeros, or none at all. So, it is important to have the more general results of Section 3.5 to characterize minimum-phase signals. In the theory of seismic deconvolution, a great deal of importance is placed on the hypothesis that impulsive seismic sources produce minimum-phase wavelets. Furthermore, the constant-Q theory of seismic attenuation (e.g., Kjartansson (1979)) predicts that a Dirac impulse injected into an anelastic medium will evolve as a minimum-phase wavelet. We have seen that a minimum-phase wavelet is completely determined by its amplitude spectrum, meaning that the phase spectrum can be calculated from the amplitude spectrum. This is significant for seismic deconvolution because the wavelet is not known a priori and must be estimated from the data itself. The estimation of the wavelet's amplitude spectrum is relatively straightforward, while the estimation of the phase spectrum is not. Thus, seismic deconvolution is enabled in the context of minimum-phase wavelets.

Exercises

3.3.3 Use z-transforms to repeat the convolution in Exercise 3.1.

3.3.4 Given a signal $[a]$, show that $a(z)z^m$ is the spectrum of the signal delayed by m samples. Thus z is the unit delay operator. What is the meaning of z^{-1}?

3.3.5 Factor each of the following signals and determine if they are minimum phase or not. Feel free to use the quadratic formula. Give the locations of the zeros for each signal. Assume the first sample is at $t = 0$.
1. $[a] = [1, 0, -0.25]$.
2. $[a] = [1, -1, 0.25]$.
3. $[a] = [1, 1.5, -1]$.
4. $[a] = [1, -1/6, -1/6]$.
5. $[a] = [1, 3/8, -7/16]$.
6. $[a] = [1, 9/14, -4/7]$.
7. $[a] = [1, 1, 1, 1]$. (Finding the zeros may not be enough to solve this one!)

3.3.6 If a signal $[a]$ is minimum phase, then will it still be minimum phase if delayed by m samples? Why? What if it is advanced by m samples?

Convolution as a Matrix Multiplication

Usually the integral of the product of two continuous functions can be approximated in a computer as a matrix–vector product. The convolution integral provides an important example. Consider the discrete convolution formula

$$c_j = \sum_{k=0}^{N-1} a_{j-k} b_k \tag{3.47}$$

and compare it with the general formula for the product of a matrix $\underline{\underline{M}}$ with a column vector $\underline{v}$, that is,

$$\underline{u} = \underline{\underline{M}}\,\underline{v}, \tag{3.48}$$

which has the equivalent component expression

$$u_j = \sum_k M_{jk} v_k, \tag{3.49}$$

where the M_{jk} are the components of $\underline{\underline{M}}$ and the v_k are the components of $\underline{v}$. (Equations (3.48) and (3.49) are two alternative notations for the same thing, with the latter being more explicit while the former is more compact. For the remainder of this book, either notation will be used as best suits the context.) Examination of Eqs. (3.37) and (3.49) shows that convolution can be done by defining a matrix $A_{jk} = a_{j-k}$. Written as a matrix operation, Eq. (3.47) becomes

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \dots & a_{-N+1} \\ a_1 & a_0 & a_{-1} & \dots & a_{-N+2} \\ a_2 & a_1 & a_0 & \dots & a_{-N+3} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N-1} & a_{N-2} & a_{N-3} & \dots & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{bmatrix}. \tag{3.50}$$

(If $[a]$ is causal, then the entries above the main diagonal of the matrix will all be zero.) This will be written as an abstract matrix equation as

$$\underline{c} = \underline{\underline{A}}\,\underline{b}, \tag{3.51}$$

where the convention is that singly underlined quantities are column vectors and a double underline denotes a matrix.

The matrix $\underline{\underline{A}}$ in Eq. (3.50) has a special symmetry that is required to perform a convolution. Though the matrix has N^2 entries, there are only N independent numbers, which all come from $[a]$. The elements along any diagonal are constant. Each column contains a copy of $[a]$ that has been shifted to place a_0 on the main diagonal. Equivalently, each row contains a time-reversed and shifted copy of $[a]$. Matrices with this symmetry are called *Toeplitz* or, sometimes, *convolution* matrices. Since convolution is commutative, it follows that forming a Toeplitz matrix with $[b]$ and a column vector with $[a]$ must give an identical result.

One way to visualize the matrix–vector multiplication in Eq. (3.50) is called *matrix multiplication by rows*. In this process, each row of $\underline{\underline{A}}$ multiplies $\underline{b}$ as a vector dot product. That is, the corresponding elements are multiplied and the resulting products are summed to a scalar that becomes the corresponding element of $\underline{c}$. This can be written as the set of equations

$$\begin{aligned}
c_0 &= a_0 b_0 + a_{-1} b_1 + a_{-2} b_2 + \ldots a_{-N+1} b_{N-1}, \\
c_1 &= a_1 b_0 + a_0 b_1 + a_{-1} b_2 + \ldots a_{-N+2} b_{N-1}, \\
c_2 &= a_2 b_0 + a_1 b_1 + a_0 b_2 + \ldots a_{-N+3} b_{N-1}, \\
\ldots &= \ldots \\
c_{N-1} &= a_{N-1} b_0 + a_{N-2} b_1 + a_{N-3} b_2 + \ldots a_0 b_{N-1}.
\end{aligned} \tag{3.52}$$

This view of matrix multiplication shows how each sample of $\underline{c}$ (each output sample) is created as a linear superposition of the inputs. An alternate but equally valid perspective comes from *matrix multiplication by columns*. Inspection of Eq. (3.52) shows that each sample of $\underline{b}$ multiplies a corresponding column of $\underline{\underline{A}}$. Thus, a column-oriented view of matrix multiplication suggests

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix} b_0 + \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ \vdots \\ a_{N-2} \end{bmatrix} b_1 + \begin{bmatrix} a_{-2} \\ a_{-1} \\ a_0 \\ \vdots \\ a_{N-3} \end{bmatrix} b_2 + \ldots \begin{bmatrix} a_{-N+1} \\ a_{-N+2} \\ a_{-N+3} \\ \vdots \\ a_0 \end{bmatrix} b_{N-1}. \tag{3.53}$$

In this view, the formation of a convolution by a scaled superposition of impulse responses (the columns of $\underline{\underline{A}}$) is manifest. The final result is obtained for all samples simultaneously by the superposition of the scaled columns of $\underline{\underline{A}}$.

Code Snippet 3.3.1 This example illustrates the convolution of a minimum-phase wavelet with a reflectivity through the construction of a convolution matrix (line 7) from the wavelet. The process is displayed in Figure 3.8a.

```
1  dt=.002;%time sample size
2  fdom=30;%dominant frequency
3  tmax=.5;%time length of reflectivity
4  tlen=.1;%time length of wavelet
5  [w,tw]=wavemin(dt,fdom,tlen);
6  [r,t]=reflec(tmax,dt,1,5,4);%reflectivity
7  W=convmtx(w,length(r));%build the convolution matrix
8  s=W*r;%perform the convolution
```

End Code

signalcode /convomat.m

Numerical Computation with Convolution Matrices

Consider the computation of a convolutional synthetic seismogram as in $s(t) = (w \bullet r)(t)$, where $w(t)$ is a wavelet and $r(t)$ is a reflectivity. MATLAB provides the command `convmtx` to allow this to be done with a convolution matrix. The matrix can be built from either r or w, but we will do the latter. Expressing this as a matrix equation, we have

$$\underline{s} = \underline{\underline{W}}\,\underline{r}, \tag{3.54}$$

where $\underline{\underline{W}}$ is a convolutional matrix built from $\underline{w}$. Proper construction of $\underline{\underline{W}}$ requires understanding its required matrix dimensions. Let M be the length of $\underline{s}$ and N be the length of $\underline{r}$. Then $\underline{\underline{W}}$ must have M rows and N columns. If the length of $\underline{w}$ is L, then the convolution length rule (Eq. (3.38)) requires $M = N + L - 1$. Thus, in addition to the vector $\underline{w}$, the construction of $\underline{\underline{W}}$ requires knowledge of N.

Code Snippet 3.3.1 is an illustration of the construction of a convolution matrix and the creation of a convolutional synthetic seismogram. In this case, the time sample size is $\Delta t = .002$ s and the wavelet length is 0.1 s so that $L = 51$. Also, the reflectivity length is 0.5 s, which means that $N = 251$ and therefore $M = 301$. Line 7 builds the convolution matrix with the command `convmtx`. The two inputs to this command are the wavelet and the length of the reflectivity to which the matrix will be applied. Thus the resulting convolution matrix has 301 rows and 251 columns. For the minimum-phase wavelet in this example, the resulting convolution matrix, $\underline{\underline{W}}$, is depicted in Figure 3.8a, where only every twentieth column is displayed. This is actually a graphical representation of Eq. (3.54) and shows $\underline{r}$ and $\underline{s}$ as well. Figure 3.8b shows a nearly identical convolution except that the wavelet this time is a zero-phase Ricker wavelet. In Figure 3.8a, $\underline{\underline{W}}$ is identically zero above the main diagonal, while in Figure 3.8b, $\underline{\underline{W}}$ is symmetric about the main diagonal. Both of these convolution matrices have the basic Toeplitz symmetry, meaning that they are constant along any diagonal. Notice also the different alignments of $\underline{r}$ in the two figures, which were used to get the correct visual alignment between $\underline{r}$ and $\underline{s}$. In both cases the seismogram is computed as $\underline{\underline{W}}\,\underline{r}$. These figures provide another perspective on how best to truncate $\underline{s}$ to

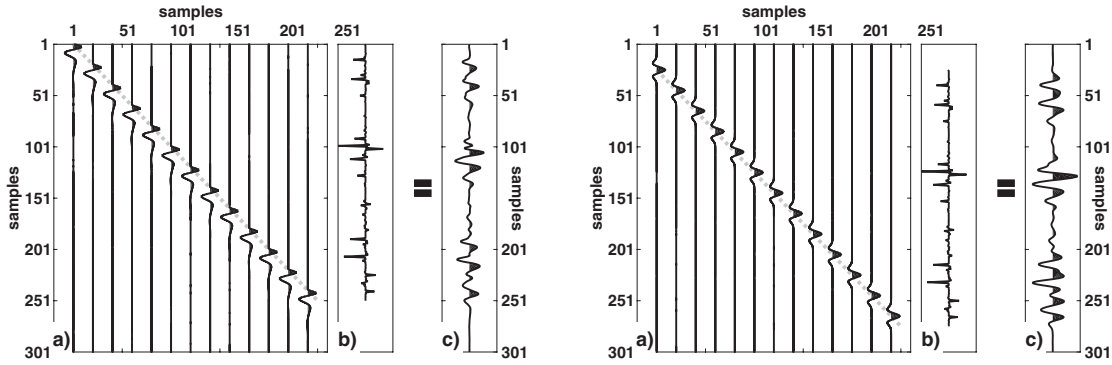


Figure 3.8a (left) Convolution of a minimum-phase wavelet with a reflectivity by matrix multiplication (Eq. (3.49)). (a) The convolution matrix (every twentieth column), where the dotted gray line denotes the main diagonal. (b) A reflectivity series being convolved with the wavelet. (c) The result of the convolution.

Figure 3.8b (right) Convolution of a zero-phase wavelet with a reflectivity by matrix multiplication (Eq. (3.49)). (a) The convolution matrix (every twentieth column), where the dotted gray line denotes the main diagonal. (b) A reflectivity series being convolved with the wavelet. (c) The result of the convolution.

the same length as $\underline{r}$. When the wavelet is causal as in Figure 3.8a, the truncation is entirely at the trailing end of $\underline{s}$. For the zero-phase case, $\underline{s}$ is shortened symmetrically by deleting samples at both ends.

Convolution by matrix multiplication is generally slower and requires more computer memory than direct implementation of Eq. (3.47). Since $\underline{\underline{W}}$ has MN samples but only L of them are unique, there is a lot of redundancy, so that storing the entire matrix is generally inefficient. However, there are cases when this is worthwhile. One of these is in the design of filters with specific properties. Here the matrix formulation facilitates well-known least-squares techniques that allow the filter's behavior to be optimized. Another instance is when the convolutional seismogram concept is extended to the anelastic case. The presence of anelasticity causes higher frequencies to attenuate more rapidly than lower frequencies, and this causes the seismic wavelet to evolve with time. The construction of a synthetic seismogram by replacing each reflection coefficient with a scaled wavelet must be done using a progressively evolving wavelet. Thus the essential translation invariance, or stationarity, of the convolution process is lost. Yet, as will be seen in the next chapter, a nonstationary convolution matrix can be constructed that is a direct generalization of the concepts here.

Convolution as a Weighted Average

Equation (3.53) shows that matrix multiplication by columns is equivalent to viewing convolution as the scaled superposition of impulse responses. However, Eq. (3.52) suggests an alternative, and equally valid, interpretation of convolution. This equation shows each element of $\underline{c}$ being formed as a weighted average of the samples of $\underline{b}$ with the rows of $\underline{\underline{A}}$

providing the weights. Of course, since convolution is commutative, $\underline{c}$ can equally well be regarded as a weighted average of $\underline{a}$ with a Toeplitz matrix $\underline{\underline{B}}$ providing the weights.

This view of convolution is valuable, since it is often desirable to smooth a physical dataset. For example, a time series that represents the measurement of a slowly changing physical quantity may fluctuate more rapidly than is easily explicable. Such fluctuations may be ascribed to contamination of the measurements by random noise. In this case, the data are often subjected to a *running average*, whereby the kth output point is formed as the average of the input points on either side of sample k. This is also called *smoothing*. The *averaging width*, or number of points involved in the average, is an important decision. For a width of m, a running average can be implemented as a convolution with a series of m ones and a division by m, or with a boxcar of length m and amplitude $1/m$.[8] For example, a running average of length 3 uses an averaging operator defined by $a_k = 1/3, -1 \leq k \leq 1$, and $a_k = 0$ otherwise. Then, $\underline{b}$ is smoothed by the matrix equation

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 1 & 0 & \dots & 0 \\ & & \dots & \dots & \dots & & \\ & & & \dots & \dots & \dots & \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \end{bmatrix}. \tag{3.55}$$

Here the convolution matrix is nonzero only on the main diagonal and the two subdiagonals above and below the main diagonal. These diagonals are filled with ones. Comparing with Eq. (3.52) shows that, for example, $c_2 = (b_1 + b_2 + b_3)/3$. Thus the conclusion is that a running average is equivalent to convolution with a properly constructed boxcar.

It is easily concluded that a running average using weights other than unity can be expressed as a convolution with an appropriate averaging function. For example, weights that ramp linearly from zero to one and back to zero can be expressed as a triangular averaging function. Multidimensional averaging works the same way; it can always be expressed as a multidimensional convolution. Later in this chapter, the connection between averaging and *low-pass filtering* will be illustrated.

Code Snippet 3.3.2 illustrates the creation of a noisy seismogram with a defined signal-to-noise ratio and then smooths this noisy trace by convolving with boxcars of increasing lengths. We will see later, after introducing the Fourier transform, that convolution with a boxcar suppresses frequencies above $f_0 = 1/(2T_0)$, where T_0 is the boxcar width in seconds. However, the rejection of higher frequencies is very imperfect and so the noise suppression results, while interesting, look as though they could be improved. In this example, the signal-to-noise ratio is 0.5, and we will see later that noise is swamping signal at about 50 Hz. With this knowledge, we might guess that the appropriate boxcar width would be 0.01 s but we see continued noise suppression all the way up to 0.02 s. It is also interesting that even the shortest boxcar has a helpful result. Careful examination (compare

[8] "Boxcar" is a common term used to refer to a rectangular pulse. It is a constant (often unity) for some defined domain, and zero elsewhere. See Figure 2.5 for examples.

Code Snippet 3.3.2 This coding example creates a noisy synthetic seismogram and then illustrates the smoothing effect of convolution with boxcars of different length. This is a type of filtering and is compared with the more sophisticated filtering available with *filtf*. The signal-to-noise ratio is 0.5, defined on line 6. The results are displayed in Figure 3.9.

```
1   %demonstrate smoothing by convolution
2   dt=.001;%time sample size
3   tmax=1;%length in seconds of reflectivity
4   fdom=30;%dominant frequency of wavelet
5   tlen=.2;%wavelet length
6   s2n=.5;%signal--to-noise ratio
7   tsmo=.004*(1:.5:5);%averaging lengths
8   [r,t]=reflec(tmax,dt,.1,5,3);%create the reflectivity
9   [w,tw]=wavemin(dt,fdom,tlen);%create the wavelet
10  s=convm(r,w);%the noise-free seismogram
11  n=rnoise(s,s2n);%create the noise to be added
12  sn=s+n;%the noisy seismogram
13  sfilt=zeros(length(s),length(tsmo)+3);%to store filtered results
14  sfilt(:,1)=s;%noise free trace in position 1
15  sfilt(:,2)=sn;%noisy trace in position 2
16  for k=1:length(tsmo)
17      nsmo=round(tsmo(k)/dt);%averaging size in samples
18      box=ones(nsmo,1);%boxcar
19      sfilt(:,k+2)=convz(sn,box/nsmo);%use convz to smooth
20  end
21  sfilt(:,k+3)=filtf(sn,t,0,[45 5]);%better low-pass filter
22
```

End Code

signalcode / conv_smoothing2.m

between 0.5 and 0.8 s the `tsmo` = 0.02 s result with the noise-free trace) shows that while the 0.02 s result has better noise suppression, the signal has also been harmed. The uppermost trace uses the more sophisticated filtering available through *filtf*, which will be discussed in more detail later in this chapter. On line 21, the *filtf* command is given as `filtf(sn,t,0,[45 5])`, where the first two arguments are just the trace to be filtered, the 0 in the third position indicates that the filter has no rejection cutoff on the low-frequency end, and the fourth argument [45 5] indicates that the filter begins rejecting high frequencies at 45 Hz (the *cutoff frequency*) and the filter "edge" is 5 Hz wide, meaning that strong rejection is achieved by 50 Hz. It seems that *filtf* has done an almost perfect job, but this is deceptive. Any attempt to broaden the spectrum beyond 50 Hz will fail on the *filtf* result, while it will work very well on the noise-free trace. This is because the noise has been suppressed by the filtering process but has not been removed. The filter applied here by *filtf* is an example of a low-pass filter. A low-pass filter has only a low-frequency cutoff, while a band-pass filter has both a low- and a high-frequency cutoff.

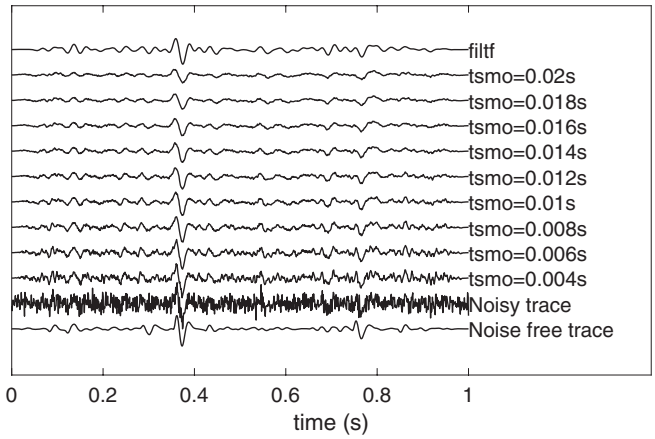


Figure 3.9 Created by Code Snippet 3.3.2, this figure shows a noise-free convolutional seismogram, a noisy version of the same, and a series of smoothed results created by convolving the noisy seismogram with boxcars (square pulses) of increasing length. The boxcar lengths are called `tsmo`. The uppermost trace is a low-pass-filtered result from the function *filtf*.

In closing this section, a brief discussion of signal-to-noise ratio, or *s*2*n*, is warranted. A basic definition of *s*2*n* is

$$s2n = \sqrt{\frac{\sum_k s_k^2}{\sum_k n_k^2}}, \tag{3.56}$$

where $\underline{s}$ is the noise-free signal and $\underline{n}$ is the pure noise. As written, Eq. (3.56) is an amplitude measure, while in some contexts a energy measure is used, which is the square of this expression. For synthetic data, *s*2*n* is relatively easy to calculate, and *rnoise* (line 11) takes $\underline{s}$ and the desired *s*2*n* as input and returns zero-mean, normally distributed random noise with the specified strength. This is then added to the noise-free signal (line 12) to produce the desired noisy signal. Such noise is said to be "white," meaning that it has the same strength at all frequencies. To estimate *s*2*n* for real data requires that we have a method of separating signal from noise. The low-pass filter applied here by *filtf* is a simple attempt at such a separation. If $\underline{sf}$ is the filtered signal, then the noise is estimated as $\underline{n} = \underline{sn} - \underline{sf}$, where $\underline{sn}$ is the original noisy signal. In the case of Code Snippet 3.3.2, using $\underline{sf}$ as the pure signal and the noise estimate just described gives $s2n \approx 0.574$, which is reasonably close to the true 0.5.

3.4 The Discrete Fourier Transform

The Fourier transform compels us to work in both the time and the frequency domains and, in order to do so in the computer, we must also address sampling in the frequency domain. This leads to the *discrete Fourier transform* and its inverse, which are direct

mappings between the samples in one domain and the samples in the other domain. Additionally, some artifacts are introduced by the sampling process. This section explores these concepts.

3.4.1 The DFT

Equation (3.18) gives the values of the Fourier transform in the interval $[-1/(2\,\Delta t), 1/(2\,\Delta t)]$. Taking N uniformly spaced samples in this interval gives a frequency sample size of $\Delta f = 1/(N\,\Delta t)$, and we can choose the sampling points as

$$f_\nu = \nu\,\Delta f, \qquad \text{for integers } \nu \text{ in the range } -\frac{N}{2} < \nu \leq \frac{N}{2}. \tag{3.57}$$

Inserting these values into Eq. (3.18), we obtain the sampled values of the Fourier transform as

$$\hat{s}(\,\nu\,\Delta f) = \Delta t \sum_k s_k e^{-2\pi i \nu\,\Delta f k\,\Delta t}. \tag{3.58}$$

In real computations on a digital computer, this infinite sum is replaced by a finite sum $\sum_{k=0}^{N-1}$, which corresponds to observing only the first N time samples $[s_0, s_1, \ldots, s_{N-1}]$ of our original signal $s(t)$. For real, causal signals, this is a good approximation provided that N is large enough, as real signals decay to zero over time. Also, the sinc function reconstruction given in Eq. (3.3) is an exact representation of a band-limited signal with these N time samples.

This finite sum suggests we define a transform from time samples to frequency samples by the formula

$$\hat{s}_\nu = \Delta t \sum_{k=0}^{N-1} s_k e^{-2\pi i \nu\,\Delta f k\,\Delta t} = \Delta t \sum_{k=0}^{N-1} s_k e^{-2\pi i \nu k/N}, \tag{3.59}$$

where in the second sum we have simplified using the identity $\Delta f\,\Delta t = 1/N$. It is customary to drop the scalar Δt in this expression to obtain the final form for the DFT,

$$\hat{s}_\nu = \sum_{k=0}^{N-1} s_k e^{-2\pi i \nu k/N}. \tag{3.60}$$

Here we have a transform from N time-domain samples s_k to N frequency-domain samples $\hat{s}_\nu$. Notice that the formula defines $\hat{s}_\nu$ for all integers ν and this sequence is periodic, for $\hat{s}_{N+\nu} = \hat{s}_\nu$ for all ν. In most numerical algorithms, the values of $\hat{s}_0, \hat{s}_1, \hat{s}_2, \ldots, \hat{s}_{N-1}$ are computed directly, with the understanding that the last half of the sequence $\hat{s}_{N/2}, \ldots, \hat{s}_{N-1}$ represents the negative-frequency samples.

The inverse transform, or iDFT, is given by the formula

$$s_k = \frac{1}{N} \sum_{\nu=0}^{N-1} \hat{s}_\nu e^{2\pi i \nu k/N}. \tag{3.61}$$

To see that Eq. (3.61) is the inverse of Eq. (3.60), change the summation index from k to j in the latter and substitute it into the former. This gives

$$s_k = \frac{1}{N}\sum_{\nu=0}^{N-1}\left[\sum_{j=0}^{N-1} s_j e^{-2\pi i\nu j/N}\right] e^{2\pi i\nu k/N}, \tag{3.62}$$

which, upon interchanging the order of summation, becomes

$$s_k = \frac{1}{N}\sum_{j=0}^{N-1} s_j \sum_{\nu=0}^{N-1} e^{2\pi i\nu(k-j)/N}. \tag{3.63}$$

The inner sum is a geometric series $\sum_{\nu=0}^{N-1} z^\nu$ with factor $z = e^{2\pi i(k-j)/N}$, so it sums to the geometric ratio $(1-z^N)/(1-z)$, which is zero, since z is an Nth root of unity, except in the case where $z = 1$ (i.e., $j = k$), in which case it sums to N. We conclude that this sum evaluates as

$$\sum_{\nu=0}^{N-1} e^{2\pi i\nu(k-j)/N} = N\delta_{jk}, \tag{3.64}$$

where δ_{jk} is the *Kronecker delta*, with the value $\delta_{jk} = 1$ if $j = k$ and $\delta_{jk} = 0$ if $j \neq k$. Therefore Eq. (3.63) evaluates as

$$s_k = \frac{1}{N}\sum_{j=0}^{N-1} s_j N\delta_{jk} = s_k. \tag{3.65}$$

Equation (3.64) is the discrete equivalent of the orthogonality relation for continuous exponentials, Eq. (2.22), and is discussed in a number of texts, including Papoulis (1984). MATLAB allows an easy numerical evaluation of Eq. (3.64) for $N = 128$, $j = 64$, and $k = 0, \ldots, 127$, which looks like

```
result=sum(exp(2*pi*i*(0:N-1)*(j-k(m))/N);
```

for the mth entry in k. Notice in this code expression that ν takes all integer values from 0 to $N-1 = 127$ and that one particular value of k, designated by the index m, is used. Placing this expression in a loop to evaluate it for all values of k allows creation of Figure 3.10. Theoretically, orthogonality is exact and a precise 0 should result for $j \neq k$. Since this computation involves the computation and summation of 128 sines and the same number of cosines, an exact zero does not result, and instead a value of about 10^{-14} is achieved. The maximum value of 128 occurs at $k = 65$ and at this point the imaginary part is precisely zero. Also, the ratio between the mean absolute value for all points where $k \neq j$ and the maximum value is 10^{-16}, which is the machine precision of the authors' computers (the MATLAB command `eps`[9] will tell you your machine precision). All of this means that the DFT is an *exact* numerical transform to the frequency domain and that the iDFT will recover the original signal to machine precision.

[9] Suppose `x=1`; then `eps` is the smallest floating-point number for which `x2=x+eps` will pass the logical test `x2>x`.

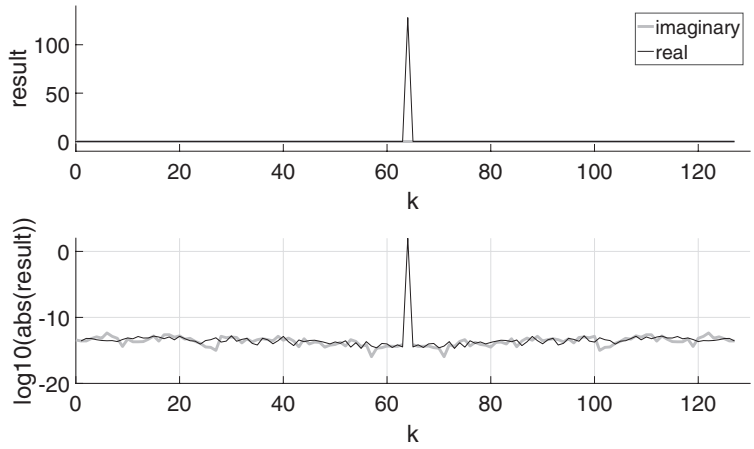


Figure 3.10 The discrete orthogonality relation, or Eq. (3.64), is demonstrated for the case of $N = 128$, $j = 64$, and $k = [0, 1, 2, \ldots, 127]$.
The value `result` was computed with `result=sum(exp(i*2*pi*(0:N-1)*(j-k)/N)`.
The upper figure plots `result` versus `k` on a linear scale, while the lower figure plots `log10(result)`. A value of 10^{-14} for `result` is essentially zero at machine precision.

3.4.2 The Fast Fourier transform

The *fast Fourier transform* is usually credited to Cooley and Tukey (1965) and is considered to be one of the most significant numerical algorithms ever invented. The reader with a deep interest in algorithms should consult one of the many texts that describe the FFT algorithm such as Bracewell (2000), as it will not be detailed here. What is important for this discussion is to understand that the FFT is not a new transform but rather a very efficient way to compute the DFT. According to most accounts, the Cooley–Tukey algorithm was a rediscovery of the efficient algorithm, with other discoveries dating back to as early as Gauss. In the first author's experience, the FFT algorithm was discovered and used by E. V. Herbert of Chevron Canada Resources in 1962. Herbert continued to maintain and expand his own library of FFT algorithms, which were proprietary to Chevron, up until his death in 1995. There were likely other such instances in other settings. The Cooley–Tukey paper was published at the right time for the emerging numerical computing industry and credit is deservedly given them. Since then, there have been many other FFT algorithms published that either expand on the original concepts or introduce new ones. Still, these are all just efficient DFTs and, for the practicing geophysicist, most of the algorithmic details are unimportant.

The most obvious way to compute the DFT from Eq. (3.60) is by a matrix–vector operation. Let $\underline{s}$ and $\underline{\hat{s}}$ be $N\times 1$ column vectors containing the time-domain and frequency-domain samples; then we can rewrite Eq. (3.60) as

$$\underline{\hat{s}} = \underline{\underline{F}}\,\underline{s}, \tag{3.66}$$

where the DFT matrix $\underline{\underline{F}}$ has components $F_{\nu k} = e^{-2\pi i \nu k/N}$, where both ν and k run over the values $0, 1, 2, \ldots, N-1$. The matrix $\underline{\underline{F}}$ is square, $N \times N$, given by

$$\underline{\underline{F}} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & e^{-2\pi i/N} & e^{-4\pi i/N} & \ldots & e^{-2\pi i(N-1)/N} \\ 1 & e^{-4\pi i/N} & e^{-8\pi i/N} & \ldots & e^{-4\pi i(N-1)/N} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & e^{-2\pi i(N-1)/N} & e^{-4\pi i(N-1)/N} & \ldots & e^{-2\pi i(N-1)^2/N} \end{bmatrix}, \tag{3.67}$$

and can be produced in MATLAB by the `dftmtx` command. In general, a square matrix times a vector is called an N^2 operation, meaning that the computation time scales as N^2. So, if 1 μs (microsecond) is required when $N = 8$, then for $N = 16$ the time required will be $16^2/8^2 = 4\,\mu s$, while for $N = 128$ it will be $128^2/8^2 = 256\,\mu s$. Thus the times increase in proportion to the square of N. The original Cooley–Tukey algorithm required that N be a power of 2, i.e., $N = 2^n$, which for $n = 1, 2, 34, 5, 6, 7, 8, 9, 10$ gives $N = 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$, and the algorithm was shown to be an $N\log(N)$ algorithm, meaning that the times increase in proportion to $N\log(N)$, not N^2. This can be a tremendous saving for large N. Again, if $N = 8$ requires 1 μs, then for values of $N = 16, 32, 64, 128, 256, 512, 1024$ an N^2 algorithm will require times of $4, 16, 64, 256, 1024, 4096$, and 16 384 μs, while an $N\log(N)$ algorithm will require times of $3, 7, 16, 37, 85, 192$, and 427 μs. It is not unusual for a modern seismic dataset to have 10^9 traces with 1000 samples each. With an N^2 algorithm having the performance hypothesized here, it will require $(16\,384)(0.000001)(10^9) \approx 1.6 \times 10^7$ s, or about 189 days, to perform a forward FFT on the entire dataset. An $N\log(N)$ algorithm will require only about 4.9 days. These are estimates for a single CPU and, when spread across a modern computing cluster with thousands of nodes, FFT-based algorithms become very attractive.

Since Cooley and Tukey (1965) there have been many advances in FFT algorithms, the most significant being the relaxation of the requirement that N be a power of 2. Modern FFTs can handle lengths that are powers of other integers, with the integer being called the *radix*, and products of powers of different integers, which are called multi-radix algorithms. There are even FFTs that can achieve $N\log(N)$ performance for any value of N, even prime numbers, and the `fft` in MATLAB is of this advanced type. Code Snippet 3.4.1 details a numerical experiment to measure the computation times of MATLAB's FFT implementation. For all array lengths N between 64 and 1024, the code computes 1000 FFTs and measures the computation time required for each length. It also computes theoretical expectations for N^2 and $N\log(N)$ performance. The results, shown in Figure 3.11, demonstrate that MATLAB's FFT achieves, and even exceeds, $N\log(N)$ performance over all measured array lengths. While the times for array lengths of 2^n are among the smallest, there are many intermediate lengths with similar performance. These results suggest that array length is not a primary consideration for efficiency when using MATLAB's FFT. The entire computation, of roughly one million FFTs with an average length of about 500 samples, required about 15 s on the first author's computer, an Intel i7-4650U device operating at 2.3 GHz.

Code Snippet 3.4.1 This example measures the computation time for MATLAB's *fft* command for all possible array lengths between 64 and 1024 (line 1). For each array length, 1000 FFTs (line 5) are computed (line 11) and the computation time is measured using the *tic* and *toc* commands (lines 9 and 13). The measured run times are compared against N^2 and $N\log(N)$ expectations, with these being calculated on lines 22–25. The results are plotted using *linesgray* and displayed in Figure 3.11.

```
1   lengths=64:1024;%signal lengths to test
2   times=zeros(size(lengths));%array for times for each length
3   l2=[64 128 256 512 1024];%power of 2 lengths
4   tl2=zeros(size(l2));%array for times of each l2 length
5   nreps=1000;%number of repititions for each length
6   t1=clock;%grab start time
7   for k=1:length(lengths)
8       s=rand(1,lengths(k));%generate a random number vector
9       tic %start time
10      for kk=1:nreps
11          S=fft(s);%here is where all the work happens
12      end
13      times(k)=toc;%grab elapsed time for nreps reps.
14      ind=find(lengths(k)==l2);%check for a power of 2 length
15      if(~isempty(ind))
16          tl2(ind)=times(k);%store result for power of 2
17      end
18  end
19  timeused=etime(clock,t1);%total time. Same as sum(times)
20  disp(['total time ' int2str(timeused) 's for '...
21      int2str(nreps*length(lengths)) 'ffts']);
22  tnln=lengths.*log(lengths);%proportional to exp. time for nlog(n)
23  tn2=lengths.^2;%proportional to expected time for n^2
24  tnln=tnln*times(100)/tnln(100);%scale tnln2 to these results
25  tn2=tn2*times(100)/tn2(100);%scale t2 to these results
26
27  hh=linesgray({lengths,times,'-',.5,0.7},{lengths,tnln,'-',.5,0},...
28      {lengths,tn2,':',.5,0},{l2,tl2,'none',.5,0,'.',6});
```

End Code

signalcode / fft_speeds .m

3.4.3 Numerical Computations of the Fourier Transform

We can learn a lot about Fourier transforms from using advanced calculus to calculate the Fourier transforms of interesting functions, but ultimately we must address numerical computations because seismic data is inherently numerical. In the previous section, MATLAB's FFT command *fft* was examined and was shown to be very advanced and efficient. However, two issues remain to be addressed in a numerical tool intended for routine use. First, it is essential to be able to calculate the frequencies for the spectrum returned by *fft*. Second, if the time series is real-valued, as all seismic traces are, then the essential symmetry induced in the Fourier transform (Eq. (2.51)) means that the negative frequencies are not

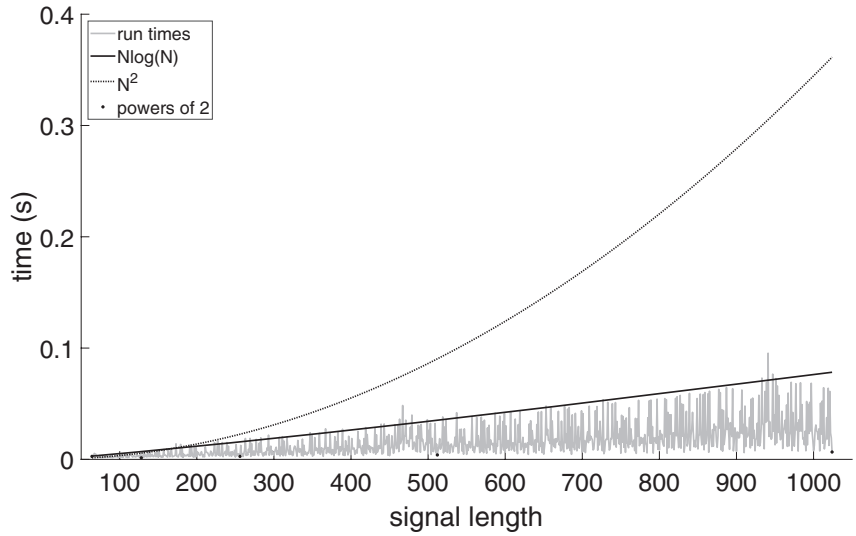


Figure 3.11 As produced by Code Snippet 3.4.1, the CPU time (measured on the first author's laptop) required to compute 1000 FFTs for all vector lengths N between 64 and 1024 is shown plotted versus N. Also shown are theoretical expectations for N^2 and $N\log(N)$ performance. Black dots indicate the times for which N is a power of 2. Roughly one million FFTs with an average length of about 500 are represented here. The total computation time was 15 s.

necessary. Returning them as part of the spectrum uses twice the computer memory that is necessary. Furthermore, a data-processing technique that operates on both positive and negative frequencies uses 100% more effort than required and, if not carefully applied, may break the symmetry of Eq. (2.51).

The easiest way to understand the frequencies for the spectrum of the DFT is to visualize them as spaced evenly around the unit circle. The DFT is given by Eq. (3.60) and computes N frequency-domain samples spaced evenly between $f = 0$ Hz and $f = 2f_{\text{nyq}}$ Hz, with the last sample being one sample before $2f_{\text{nyq}}$ because $2f_{\text{nyq}}$ is the same as 0 Hz. This means that the frequency sample size is

$$\Delta f = \frac{2f_{\text{nyq}}}{N} = \frac{1}{N\,\Delta t}. \tag{3.68}$$

There are many time–frequency dualities, and one of these is with the sample sizes in each domain. From the above equation, we see that Δf= 1/(time length of signal). On the other hand, $\Delta t = 1/(2f_{\text{nyq}})$ is essentially $\Delta t = 1$/(frequency length of signal). This frequency-domain sampling gives the values $f_\nu = 0, \Delta f, 2\,\Delta f, \ldots, \nu\ Df, \ldots, (N-1)\,\Delta f, N\,\Delta f$, where any $f_\nu > f_{\text{nyq}}$ is actually a sample at the negative frequency $f_- = -2f_{\text{nyq}} + f_\nu$ on the first alias to the right in Figure 3.3b. This infinitely periodic nature of the sampled spectrum and the locations of the f_ν can be understood as a mapping to the unit circle through the z-transform. This maps $f = 0$ to the point $z = 1$ on the positive real axis and $f = f_{\text{nyq}}$ to $z = -1$ on the negative real axis. The positive frequencies are found on the upper half of

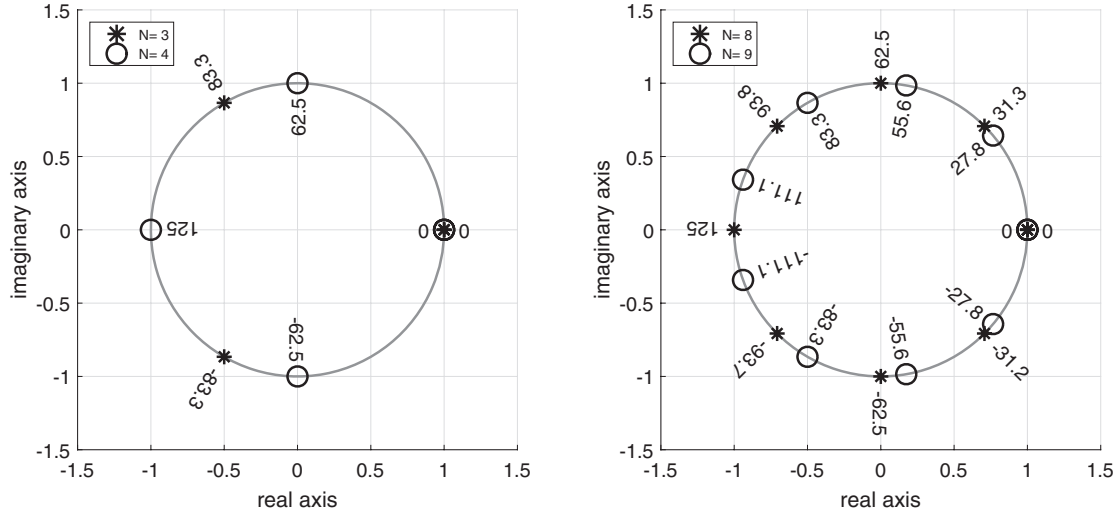


Figure 3.12a (left) The locations of frequency samples computed by the DFT for the cases $N = 3$ and $N = 4$. The sample locations are labeled with their frequencies (in Hz) for the case $\Delta t = 0.004$ s. Only when N is even is a sample obtained at $f_{\text{nyq}} = 125$ Hz.

Figure 3.12b (right) Similar to Figure 3.12a except that $N = 8$ and $N = 9$.

the unit circle and the negative frequencies on the lower half. The locations of the DFT frequency samples and their frequency values are shown as points on the unit circle in Figure 3.12a for $N = 3$ and $N = 4$, and Figure 3.12b does the same for $N = 8$ and $N = 9$. It is apparent that only when N is even is a sample obtained at f_{nyq}, while there is always a sample at $f = 0$.

MATLAB's `fft` command returns the signal spectrum with the frequencies in the natural order $f_\nu = \nu\,\Delta f, \nu \in [0, 1, 2 \ldots, N-1]$, and thus the first half of the samples are the positive frequencies and the last half are the negative frequencies. In this order, the spectrum is said to be *wrapped*. The command `fftshift` converts a wrapped spectrum into a *centered* spectrum in which 0 Hz is in the middle. MATLAB does not provide a utility to compute the frequencies of the spectral samples, but there is such a tool in the *NMES Toolbox* called `freqfft`.

The DFT takes N time-domain samples, s_k, into N frequency-domain samples, $\hat{s}_\nu$. However, the $\hat{s}_\nu$ are complex numbers, while the s_k are usually real numbers. Since each complex number is actually two real numbers (the real and imaginary parts), it seems that we are actually taking N real numbers into $2N$ real numbers. This implies that the $N\,\hat{s}_\nu$ values are not all independent. In fact, we have seen that the Fourier transform of a real signal results in a spectrum with Hermitian symmetry (Eq. (2.51)), and the same thing is true for the DFT. Hermitian symmetry says that $\hat{s}^*_\nu = \hat{s}_{-\nu}$, meaning that the negative frequencies are the complex conjugates of the positive ones. Two frequencies, 0 and f_{nyq}, are always real and so can be considered as either positive or negative frequencies. We will include

Code Snippet 3.4.2 This code creates a convolutional synthetic seismogram, `s`, from a wavelet, `w`, and a synthetic reflectivity, `r`, and then computes the Fourier transforms of all three components. For `s`, the DFT is computed as one-sided (positive frequencies only) using *fftrl* on line 13, and two-sided using *fft* on line 14. The two-sided spectrum on line 14 is wrapped but it is unwrapped by *fftshift* on line 16. Frequency coordinate vectors for the two-sided spectra are computed by *freqfft* on lines 15 and 17. The results are plotted in Figure 3.13a. `w` and `r` have only one-sided spectra computed, and the time-domain and frequency-domain views are shown in Figure 3.13b.

```
1   dt=.002;%time sample size
2   tmax=1;%maximum time
3   fdom=30;%wavelet dominant frequency
4   tlen=.3;%wavlet length
5   [r,tr]=reflec(tmax,dt,.2,3,4);%make a reflectivity
6   [w,tw]=wavemin(dt,fdom,tlen);%wavelet
7   s=conv(r,w);%use conv not convm to avoid truncation effects
8   t=(0:length(s)-1)*dt;%time coordinate for s
9   fnyq=.5/dt;%Nyquist frequency
10
11  [R,fr]=fftrl(r,tr);%one-sided spectrum of r
12  [W,fw]=fftrl(w,tw);%one-sided spectrum of w
13  [S,f]=fftrl(s,t);%one-sided spectrum of s
14  S2w=fft(s);%two sided spectrum of s (wrapped)
15  f2w=freqfft(t,length(s),1);%frequency coordinate for S2w
16  S2=fftshift(S2w);%two-sided spectrum unwrapped
17  f2=freqfft(t);%frequency coordinate for S2
```

End Code

signalcode / fft_and_fftrl .m

them with the positive frequencies. When Hermitian symmetry is considered, only the positive frequencies are required, since the negative frequencies are determined by them. Thus, the DFT maps N real numbers into N real numbers or into $2N$ complex numbers with a Hermitian symmetry linking one half of them to the other half. When dealing with large data volumes, it is wasteful to store all N complex numbers of $\hat{s}_\nu$ and, when processing is done in the frequency domain, it is pointless to process both positive and negative frequencies. Such processing must be done carefully to preserve Hermitian symmetry so that a real signal will result after inverse transformation. However, it is much more efficient and sensible to process only the positive frequencies and deduce the negative frequencies by Hermitian symmetry. This more efficient approach is enabled by *fftrl*, which calls MATLAB's *fft* but returns only the positive frequencies. Also, *fftrl* requires two inputs, which are `s`, the signal, and `t`, the time coordinates of the signal, and returns two outputs, being `S`, the spectrum (positive frequencies), and `f`, the frequency coordinates of the spectrum. In contrast, *fft* takes one input and returns one output, leaving the user to compute the frequency coordinates and separate the positive and negative frequencies. The inverse of *fftrl* is accomplished by *ifftrl*. It is a mistake to use *fftrl* for complex-valued time series.

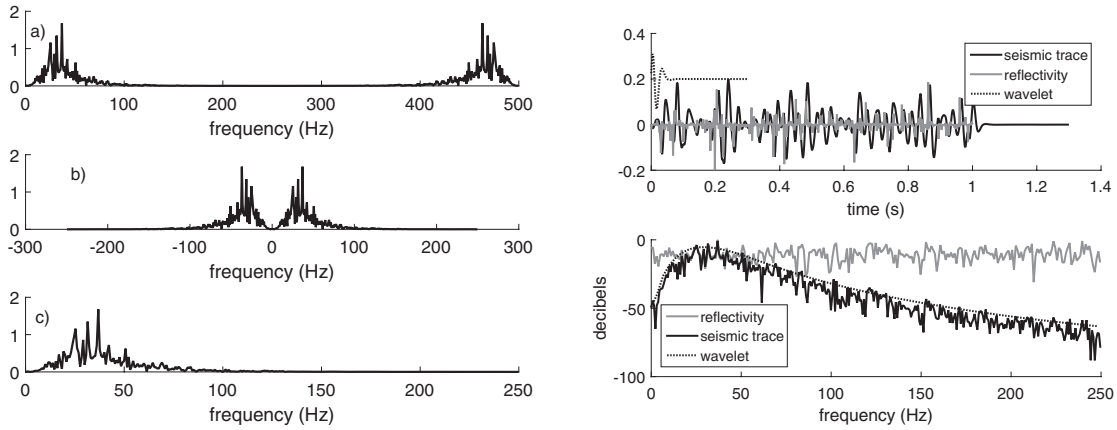


Figure 3.13a (left) For a time series s sampled at $\Delta t = 0.002$ s, the DFT amplitude spectra are shown in three different formats. (a) The two-sided spectrum as computed by *fft* is shown in wrapped format. Frequencies greater than $f_{nyq} = 250$ Hz are actually negative frequencies. (b) The same spectrum as in panel a except that it is unwrapped, or centered. Negative frequencies are properly identified. (c) The spectrum of the same signal as computed by *fftrl*. Only the positive frequencies are present.

Figure 3.13b (right) A convolutional seismic trace is shown in both the time domain (top) and the frequency domain (bottom).

As an illustration of these DFT computations, Code Snippet 3.4.2 shows the creation of a simple convolutional seismogram $s(t) = (w \bullet r)(t)$ and then computes the DFT of the wavelet (w), reflectivity (r), and seismogram (s). For the seismogram, the DFT is computed in three ways: (1) as a two-sided, wrapped spectrum using *fft*, (2) as a two-sided, unwrapped spectrum using *fftshift* after *fft*, and (3) as a one-sided spectrum using *fftrl*. The amplitude spectra are shown in Figure 3.13a plotted versus their frequencies. The frequency coordinate for the two-sided spectra comes from *freqfft*, while for the one-sided spectrum it is the second return from *fftrl*. Once the spectra are computed, the three signals (w, r, and s) are shown in both the time domain and the frequency domain in Figure 3.13b. Here the amplitude spectra have been converted to decibels by using *todb* so that a greater dynamic range can be visualized.

Although not demonstrated here, *fftrl* can be inverted by *ifftrl*, but there is one potential ambiguity. As shown in Figure 3.12b, both $N = 8$ and $N = 9$ have the same number (five) of positive frequencies (including the samples at 0 and f_{nyq} if present). This ambiguity exists for any even N and the next larger integer. An even N has $N/2+1$ samples in its one-sided spectrum, while an odd N has $floor(N/2) + 1$ samples. Therefore, when *ifftrl* is reconstructing the two-sided spectrum for passage to *ifft*, it must determine whether the two-sided spectrum was for an even or an odd N. It does so by using the fact that the sample at f_{nyq} must be entirely real. Therefore if the last sample in the one-sided spectrum is found to be complex, the inverse transform will result in an odd number of samples. This can cause a problem when data processing has been conducted in the frequency domain. Care must be taken to ensure that the processing is done in such a way that the sample at f_{nyq}, if present, remains real-valued.

Exercises

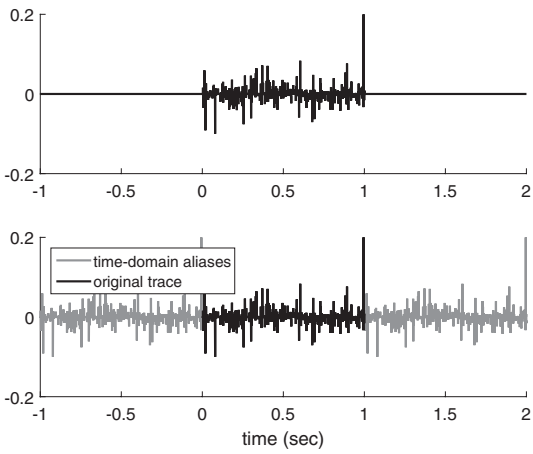
3.4.1 Show that the sample at $f = f_{\text{nyq}}$ from the DFT for N even must be real-valued. In fact, show that it is given by $\hat{s}_{\nu=N/2} = \sum_{k=0}^{N-1} s_k(-1)^k$.

3.4.2 Load the real seismic shot record `smallshot.mat` and use `fftrl` to compute the Fourier transform of the entire shot (see Figure 1.10b on page 21). (You don't need to write a loop, as `fftrl` is vectorized to automatically transform the columns of a matrix and return a frequency-domain matrix with the same number of columns.) Display both the amplitude and the phase spectra using `plotimage`. Write a script to do this. Compute the spectrum of the entire shot record, and then for an offset dependent window, 0.4 s wide, that begins just above the first break and follows the first break with increasing offset. Simple boxcar windowing will suffice. Compute the spectrum in a second window, again 0.4 s wide, that is constant with offset and begins at 2.0 s. Compare the three amplitude spectra and explain how and why they are different.

3.4.4 Time-Domain Aliasing

The development of the DFT introduced sampling in the frequency domain. The effects of this sampling can be analyzed in exactly the same way that time-domain sampling was studied previously. Thus, we envision multiplying $\hat{s}_{\Delta t}(f)$ by a sampling comb that selects N samples distributed between $f = 0$ and $f = 1/\Delta t$ with a spacing of $\Delta f = 1/(N\,\Delta t)$, and the corresponding operation in the time domain becomes convolution with a time-domain comb having a spacing of $(\Delta f)^{-1} = N\,\Delta t$. So, frequency-domain sampling causes the time-domain signal to become replicated at intervals of $N\,\Delta t = T$, where T is the length of the original signal. This is called *time-domain aliasing*, or sometimes temporal wraparound. It is important to consider time-domain aliasing when exploiting the convolution theorem to apply a filter using the DFT.

According to the convolution theorem, the time-domain convolution $s_t(t) = (r \bullet w)(t)$ should be the same thing as the frequency-domain multiplication $s_f(t) = F^{-1}[\hat{r}\hat{w}](t)$, where $F^{-1}[\cdot]$ indicates the inverse Fourier transform. When the DFT/IDFT pair are used as the Fourier tools, then unexpected results can occur if time-domain aliasing is not considered. In Figure 3.14a, the time-domain periodicity induced by frequency-domain sampling is illustrated. This reflectivity was created by Code Snippet 3.4.3, and a large spike has been deliberately placed near $t = 1$ and is intended to cause obvious problems when a minimum-phase wavelet is used in the computation of $s_f(t) = F^{-1}[\hat{r}\hat{w}](t)$. In the bottom panel of Figure 3.14b, the result of a time-domain convolution between the reflectivity of Figure 3.14a and the minimum-phase wavelet (upper panel of Figure 3.14b) is compared with the result from frequency-domain multiplication. As can be seen, there is disagreement at early times, which is caused by time-domain aliasing. Both $s_t(t) = (r \bullet w)(t)$ and $s_f(t) = F^{-1}[\hat{r}\hat{w}](t)$ can be visualized by a convolution-by-replacement process (see Section 2.3.1), but different reflectivities must be used. For s_t, use the reflectivity in the top



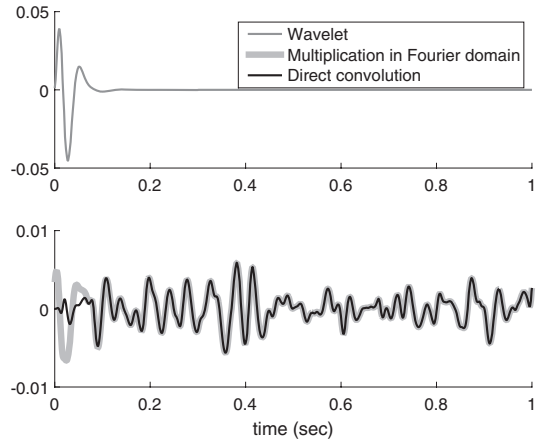


Figure 3.14a (left) Top: A 1 s long reflectivity series shown on a segment of the real line. A large spike has been deliberately placed near $t = 1$ s. Bottom: The same reflectivity series with its first time-domain aliases on either side. These aliases are induced by frequency-domain sampling and must be considered whenever the DFT is used.

Figure 3.14b (right) Top: The wavelet used to convolve with the reflectivity after padding with zeros to the same length as the reflectivity. Bottom: A comparison between time-domain convolution and frequency-domain multiplication of the spectra of the wavelet in the upper panel with the reflectivity of Figure 3.14a. The disagreement at early times is caused by time-domain aliasing. See Code Snippet 3.4.3 for computation details.

panel of Figure 3.14a and it is obvious that there is no way that the large spike at the end of the reflectivity should affect the s_t at early times. However, for s_f the reflectivity in the bottom panel of Figure 3.14a must be used, and it then becomes clear that the large spike at the end of the left alias is adjacent to the early samples of the primary r. Then, replacing the large spike with the minimum-phase wavelet allows the left alias to contaminate the early times of the convolution. (The computation code is in Code Snippet 3.4.3.)

The solution to this problem is to pad (or extend) the reflectivity with zeros to a length sufficient to avoid the wraparound problem. The required zero-pad must be at least as long as the wavelet. In Code Snippet 3.4.3, this is done with the `pad_trace` command, which has the action of extending the first input with zeros to the length of the second input. This command is actually used in several places for different purposes. In the first instance, on line 6, the wavelet, which was created 0.3 s long, is extended in length with a zero pad to match the 1 s length of the reflectivity. This is necessary to do frequency-domain multiplication because the spectra must be arrays of exactly the same size in order to be multiplied with the `.*` operator (line 7). The padding in this instance does not solve the temporal aliasing; it merely enables the spectral multiplication. It is the padding of the reflectivity on line 9 that eliminates the temporal aliasing. Subsequently, the wavelet must be padded again because the length of the reflectivity has changed. The result of both convolutions after padding the reflectivity is shown in Figure 3.15. The convolutions are now identical except that the time-domain result is shown truncated to the original 1 s length of the reflectivity, while the frequency-domain result is shown without truncation.

Code Snippet 3.4.3 This code demonstrates the use of the DFT to perform a convolution by multiplication in the frequency domain. Lines 1–3 create a random reflectivity and place a large spike at the end to cause obvious time-domain aliasing. Line 5 convolves the reflectivity with a minimum-phase wavelet generated on line 4. In order to perform a frequency-domain multiplication, the wavelet is first extended to the same length as the reflectivity using *pad_trace* on line 6. The frequency-domain multiplication and inverse Fourier transformation are on line 7. The results are shown in Figures 3.14a and 3.14b. To prevent time-domain aliasing, the reflectivity must first be padded with a sequence of zeros at least as long as the wavelet (line 9). Then lines 11 and 12 repeat the frequency-domain multiplication with the padded reflectivity, and the results are shown in Figure 3.15.

```
1  dt=.001;tmax=1;fdom=20;tlen=.3;
2  [r,t]=reflec(tmax,dt,.1,3,pi);%make a synthetic reflectivity
3  r(end-5)=.2;%insert a large spike near the end of the trace
4  [w,tw]=wavemin(dt,fdom,tlen);% a minimum phase wavelet
5  s_td=convm(r,w);%time domain convolution
6  wimp=pad_trace(w,r);%pad wavelet with zeros to length of r
7  s_fd=ifft(fft(r).*fft(wimp));%frequency domain multiplication
8  %to avoid time-domain aliasing, apply a zero pad before filtering
9  rp=pad_trace(r,1:1301);%this applies a 300 sample zero pad
10 tp=.001*(0:1300);%t coordinate for rp
11 wimp2=pad_trace(w,rp);%pad wavelet with zeros to length of rp
12 s_fdp=ifft(fft(rp).*fft(wimp2));%frequency domain multiplication
```

End Code

signalcode / time_domain_aliasing_central2 .m

Given that a convolution can be done with an FFT, it is natural to wonder which is faster. Code Snippet 3.4.4 is an experiment to test the speed of time-domain convolution, $(s_1 \bullet s_2)(t)$, versus frequency-domain multiplication, $F^{-1}[\hat{s}_1 \hat{s}_2]$. The computation time is measured for all array lengths N between 4 and 1024. Both "long" and "short" convolutions are done, where "long" means that both s_1 and s_2 have length N while "short" means that s_1 has length N while s_2 has length $round(N/4)$. For the FFT case, both arrays are necessarily of length N. For each case and each array length, 1000 repetitions are done and the computation time is measured using the *tic* and *toc* facility. The results are shown in Figure 3.16.

Exercises

3.4.3 What is the matrix $\underline{\underline{F^{-1}}}$ that performs the inverse DFT? Express it in terms of the forward DFT matrix. Show that $\underline{\underline{F^{-1}}}\,\underline{\underline{F}} = \underline{\underline{F}}\,\underline{\underline{F^{-1}}} = \underline{\underline{I_N}}$, where $\underline{\underline{I_N}}$ is the $N \times N$ identity matrix.

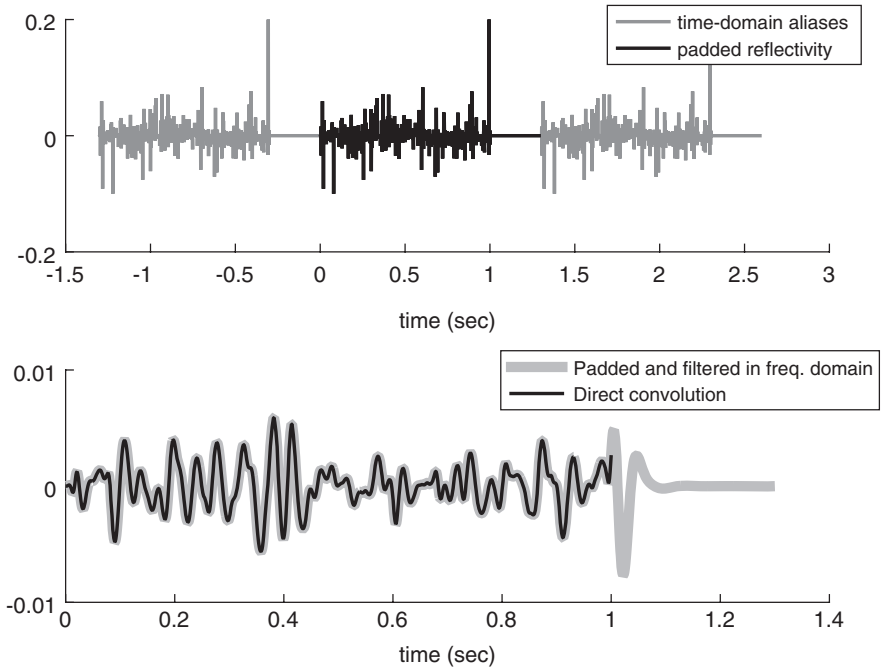


Figure 3.15 Top: The time-domain aliasing caused by frequency-domain sampling when the reflectivity has a zero pad attached. Compare with Figure 3.14a. Bottom: The results of a frequency-domain multiplication between the spectra of the padded reflectivity and the wavelet are compared with a time-domain convolution. Compare with Figure 3.14b. See Code Snippet 3.4.3 for computation details.

3.5 Discrete Minimum Phase

As noted in the previous chapter, impulsive sources such as dynamite blasts, weight drops, or earthquakes are characterized by the minimum-phase property of concentrating energy near the start of the signal. We carry over this idea to discrete-time, sampled signals with the following definition:

Discrete minimum phase A causal, sampled signal $s_0, s_1, s_2, \ldots$ is said to be *discrete minimum phase* if it maximizes the energy in the first few samples $s_0, s_1, \ldots, s_n$ compared with any other causal signal $r_0, r_1, r_2, \ldots$ with the same amplitude spectrum. That is,

$$\sum_0^n |r_k|^2 \leq \sum_0^n |s_k|^2 \text{ for any } n \geq 0, \text{ and any signal } r_0, r_1, r_2, \ldots \text{ with } |\hat{r}| \equiv |\hat{s}|.$$

This definition becomes a computational tool for finding minimum-phase signals by some results from complex analysis. An excellent reference for the relevant mathematics

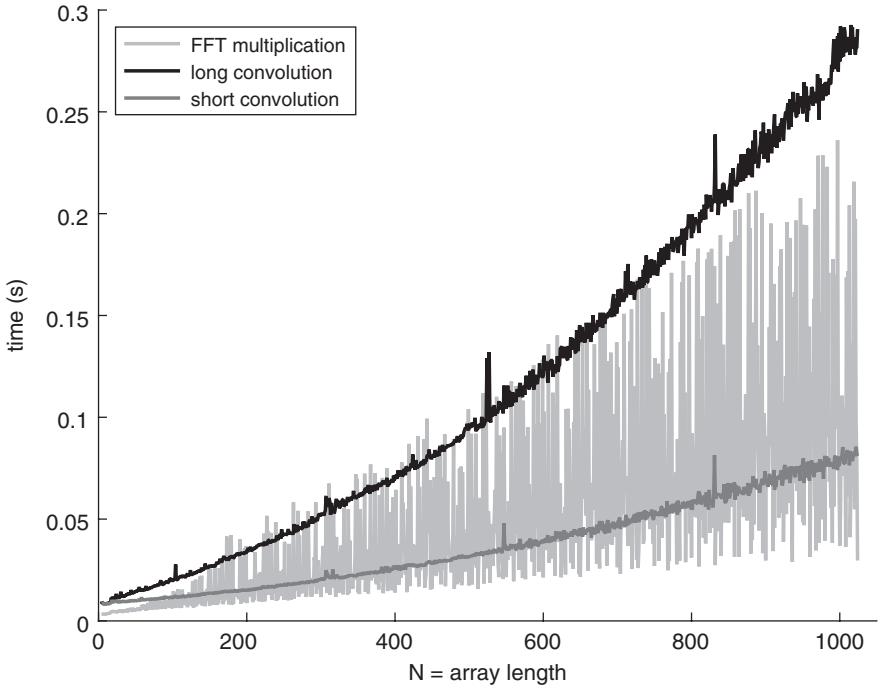


Figure 3.16 The time taken for 1000 repetitions of time-domain convolution versus Fourier-domain multiplication is shown versus array length, N, for lengths from 4 to 1024. "Long" convolution refers to the case when both arrays are of length N, while in "short" convolution one array is of length N and the other is of length $N/4$. For the FFT computation, both arrays must be of length N. See Code Snippet 3.4.4 for the computations.

is contained in Hoffman (1962). We begin with the connection to outer functions on the unit disk in the complex plane:

Theorem *A causal, sampled signal $s_0, s_1, s_2, \ldots$ is minimum phase if and only if the complex analytic function $S(z) = \sum_{k=0}^{\infty} s_k z^k$ is given as an integral of its log amplitude on the unit circle:*

$$S(z) = \lambda \exp\left(\int_{-1/2}^{1/2} \ln |S(e^{2\pi i\theta})| \frac{e^{2\pi i\theta} + z}{e^{2\pi i\theta} - z}\, d\theta \right), \tag{3.69}$$

for all z in the unit circle, and some fixed complex constant λ. In other words, the function $S(z)$ on the unit disk in the plane is an outer function in the sense of complex variables.

A moment's calculation from the definition of $S(z)$ as a sum of powers of z shows that

$$S(e^{-2\pi i\theta}) = \sum_k s_k e^{-2\pi ik\theta} = \sum_k s_k e^{-2\pi ik(f\Delta t)} = \frac{\hat{s}(f)}{\Delta t}, \tag{3.70}$$

Code Snippet 3.4.4 A code designed to compare the time to do a direct convolution in the time domain versus an FFT multiplication. All possible array lengths from 4 to 1024 are tested (line 1). For each array length, 1000 repetitions of long convolutions, short convolutions, and Fourier-domain multiplication are performed. Long convolution refers to convolving two signals together when both are of length `N` (lines 12–14). Short convolution uses one array of length `N` and another of length `round(N/4)` (lines 17–19). The FFT method is on lines 22–24. The results are shown in Figure 3.16.

```
1   N=4:1024;
2   nreps=1000;
3   tc=zeros(size(N));
4   tc2=zeros(size(N));
5   tfft=zeros(size(N));
6
7   for k=1:length(N)
8       s1=rand(1,N(k));
9       s2=s1;
10      ss2=s2(1:round(length(s2)/4));
11      tic
12      for kk=1:nreps
13          tmp=conv(s1,s2);
14      end
15      tc(k)=toc;
16      tic
17      for kk=1:nreps
18          tmp=conv(s1,ss2);
19      end
20      tc2(k)=toc;
21      tic
22      for kk=1:nreps
23          tmp=ifft(fft(s1).*fft(s2));
24      end
25      tfft(k)=toc;
26      if(rem(k,20)==0)
27          disp(['Finished k=' int2str(k)])
28      end
29  end
```

End Code

signalcode / conv_vs_fft .m

where the argument in the exponents has $\theta = f\Delta t$, by Eq. (3.18). From this we conclude that the analytic function $S(z)$ takes values on the unit circle $|z| = 1$ given by the Fourier transform $\hat{s}$ on its frequency interval. We can think of $S(z)$ as the function obtained by wrapping $\hat{s}(f)$ around the unit circle, and extending analytically to the interior of the circle.

To verify the theorem above, suppose a causal signal $r_0, r_1, r_2, \ldots$ has the same amplitude spectrum as the signal $s_0, s_1, s_2, \ldots$ satisfying the outer-function condition. Using

the analogous extension $R(z) = \sum_k r_k z^k$ to define an analytic function on the disk, we obtain

$$R(z) = S(z)G(z), \text{ for all } z \text{ in the unit disk,} \tag{3.71}$$

where the ratio $G(z) = R(z)/S(z)$ is analytic on the disk, since the denominator $S(z)$ has no zeros. The function $G(z)$ has an amplitude $|G(e^{2\pi i\theta})|$ equal to one on the unit circle, since the ratio of Fourier amplitudes $|\hat{r}(f)| \equiv |\hat{s}(f)|$ will be one. The function identity $G(z)S(z) = R(z)$ is the inner/outer factorization of $R(z)$, where $G(z)$ is the inner function and $S(z)$ is the outer function.[10] Since the function $G(z)$ is analytic and bounded by one on the unit circle, we know that $|G(0)| \leq 1$. Thus

$$|r_0| = |R(0)| = |S(0)| \cdot |G(0)| \leq |S(0)| = |s_0|. \tag{3.72}$$

That is, we have shown that the signal $s_0, s_1, s_2, \ldots$ maximizes the energy at the first sample s_0.

A similar calculation with the polynomial $S_n(z) = \sum_{k=0}^{n} s_k z^n$ and factoring through $G(z)$ will show the inequality

$$\sum_0^n |r_k|^2 \leq \sum_0^n |s_k|^2,$$

for any integer n, as desired. That is, an outer function results in a minimum-phase signal.

Conversely, suppose the signal $s_0, s_1, s_2, \ldots$ is minimum phase. In particular, by definition it maximizes energy at the first sample s_0 of all signals with the same amplitude spectrum. Define a new analytic function $R(z)$ by the formula

$$R(z) = \exp\left(\int_{-1/2}^{1/2} \log|S(e^{2\pi i\theta})| \frac{e^{2\pi i\theta} + z}{e^{2\pi i\theta} - z}\, d\theta\right), \tag{3.73}$$

for all z in the unit circle. Then $|R(z)| = |S(z)|$ on the unit circle (since their log amplitudes are the same there), and $|R(0)| = |r_0| \leq |s_0| = |S(0)|$ since the signal $s_0, s_1, \ldots$ maximizes energy at the first sample. Thus the analytic function $S(z)/R(z)$ is bounded by 1 on the unit circle, and is greater than or equal to 1 at the center of the disk. By analyticity, it must be a constant, of magnitude one. Consequently, $S(z)$ is a constant times the outer function $R(z)$, so $S(z)$ itself is outer. This completes the proof of the theorem.

In signal theory, one often encounters the phrase "a stable, causal filter with a stable causal inverse is minimum phase." Sampled signals are more general than filters, but an analogous result is true, which we state as follows:

Theorem *Suppose a causal signal $s_0, s_1, s_2, \ldots$ has a convolutional inverse $r_0, r_1, r_2, \ldots$ which is also causal and a finite-energy signal. That is, the discrete convolution of r and s yields $r \bullet s = (1, 0, 0, 0, \ldots)$. Then the signal $s_0, s_1, s_2, \ldots$ is minimum phase.*

[10] An inner function is defined as an analytic function $G(z)$ on the disk with $|G(z)| \leq 1$ everywhere.

To see this result, we use the inner/outer factorization for complex functions on the disk. Defining $S(z) = \sum_k s_k z^k$ and $R(z) = \sum_k r_k z^k$, the convolutional identity gives the product $S(z)R(z) = 1$. Writing $S(z) = G(z)H(z)$ as an inner/outer factorization, we observe that $R(z) = 1/(G(z)H(z))$ is a product of two reciprocals $G^{-1}(z)H^{-1}(z)$, and since the reciprocal G^{-1} of an outer function is also outer, this is also the inner/outer factorization for $R(x)$, with $H^{-1}(z)$ the inner part. However, if both $H(z)$ and $1/H(z)$ are inner, then they are both bounded by one on the disk, from which we conclude that $|H(z)| \equiv 1$ on the disk, and so it must be a constant. Thus the function $S(z)$ is equal to the outer function $G(z)$, times a constant, so $S(z)$ is itself outer. By definition, the signal $s_0, s_1, s_2, \ldots$ is minimum phase.

A note of caution, though. Any causal signal $s_0, s_1, s_2, \ldots$ with $s_0 \neq 0$ has a causal, convolutional inverse: this is a simple algebraic computation. The important condition in the previous theorem is that this inverse have finite energy. Also, the theorem only goes in one direction: a signal with a bounded inverse is minimum phase, but it is not always the case that a minimum-phase signal has a bounded inverse.

A useful computational tool is to observe that the phase and amplitude of the Fourier transform of a minimum-phase signal are related via the discrete version of the Hilbert transform. If a causal signal $s_0, s_1, s_2, \ldots$ is minimum phase, then the analytic function $S(z) = \sum_k s_k z^k$ is never zero in the unit disk, and so its logarithm $L(z) = \log(S(z))$ is also analytic in the unit disk. The real and imaginary parts

$$L_R(z) = \log|S(z)|, \qquad L_I(z) = \arg(S(z)) \tag{3.74}$$

are conjugate harmonic functions, and so their boundary values on the unit circle form (circular) Hilbert transform pairs. That is,

$$\arg(S(e^{2\pi i\theta})) = \int_0^1 \log|S(e^{2\pi i\theta'})| \cot(\pi(\theta - \theta'))\, d\theta', \tag{3.75}$$

$$\log|S(e^{2\pi i\theta})| = -\int_0^1 \arg(S(e^{2\pi i\theta'})) \cot(\pi(\theta - \theta'))\, d\theta', \tag{3.76}$$

where these integrals over the kernel $\cot(\pi(\theta - \theta'))$ define the circular Hilbert transform.

Writing the Fourier transform of the signal in amplitude/phase form,

$$\hat{s}(f) = A_s(f)e^{i\phi_s(f)} \quad \text{for} \quad -\frac{1}{2\,\Delta t} \leq f \leq \frac{1}{2\,\Delta t}, \tag{3.77}$$

we again identify the amplitude and phase at frequency f with the boundary values of $S(z)$ as

$$A_s(f) = |S(e^{-2\pi i f\,\Delta t})|, \qquad \phi_s(f) = \arg(S(e^{2\pi i f\,\Delta t})). \tag{3.78}$$

We can think of this as rescaling $f \mapsto f/\Delta t = \theta$ and renormalizing the frequencies to a unit interval $[-\frac{1}{2}, \frac{1}{2}]$.

Code Snippet 3.5.1 An example of computing a minimum-phase version of a sampled signal. We use the circular Hilbert transform to compute the phase spectrum from the amplitude spectrum of the initial signal. It is useful to zero pad the source in order to get a smooth interpolation of the amplitude spectrum, to obtain a stable FFT of its logarithm. The result is shown in Figure 3.17.

```
% Piece of an exponential ramp
Fs = 1024;
t = linspace(0,1,Fs);
x = exp(t).*(t>.4).*(t<.7);
% zero pad for interpolated amplitude spectrum
xlen = length(x);
xf=fft([x,zeros(1,31*xlen)]);
A=abs(xf);
% Compute the phase via Hilbert transform
Ph = ifft(log(max(A,.001*max(A)))); % stability factor .001
n = length(Ph);
Ph([1 n/2+1]) = 0;   % zero out DC and Nyquist
Ph((n/2+2):n) = - Ph((n/2+2):n);  % flip signs for Hilbert
Ph = fft(Ph);
% construct the minimum phase version
xfmin = A.*exp(Ph);
xmin=real(ifft(xfmin));
xmin=xmin(1:xlen);   % truncate back to original size
plot(t,x,'--k',t,xmin,'-k')
```

End Code

signalcode / tomino.m

We summarize this characterization as follows:

Theorem *If a causal signal $s_0, s_1, s_2, \ldots$ is minimum phase, then the (renormalized) log amplitude spectrum and phase spectrum form a Hilbert transform pair. That is to say, the real and imaginary parts of* $\log \hat{s}(\theta/\Delta t)$ *are Hilbert transforms of each other.*

As noted in the previous chapter, in many geophysical situations, we often have good estimates for amplitude spectra, but not phase. These results provide algorithms for practical methods to compute phase spectra for minimum-phase signals.

In summary, we have three useful characterizations of a discrete-time minimum-phase signal: (i) it is a causal signal which has the energy maximized near the initial sample s_0; (ii) its Fourier spectrum extends to an outer function on the unit disk; and (iii) its log amplitude spectrum and the phase spectrum form a circular Hilbert transform pair. The phase spectrum defined by Eq. (3.75) is given the special name of the minimum-phase spectrum. The physical intuition is that the minimum-phase signal is the most *front-loaded* signal possible that both is causal and has the given amplitude spectrum. Computationally, we can find the minimum-phase signal from its amplitude spectrum, using either the outer-function formula or the circular Hilbert transform. Code Snippet 3.5.1 gives an example of converting a signal to its minimum-phase version, with the results shown in Figure 3.17.

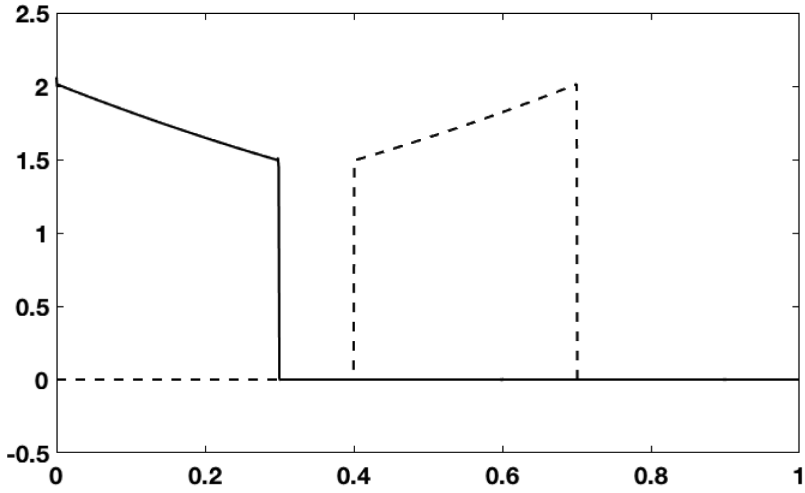


Figure 3.17 An exponential ramp in the range $0.4 < t < 0.7$ and its minimum-phase equivalent. Note the minimum-phase signal starts at $t = 0$ and is exponentially decaying.

3.6 Filtering and Spectral Analysis

3.6.1 Band-Pass, High-Pass, and Low-Pass filters

We have already encountered filtering as a convolutional process; here, we examine a variety of approaches to filtering to isolate specific frequency bands. There are three common filter types for this purpose: band-pass, high-pass, and low-pass. A fourth filter type, band-stop, is less common and will not be discussed. The terms band-pass, high-pass, and low-pass all refer to the amplitude spectrum of the filtering process, while the phase spectrum is usually either zero or minimum. Zero phase is most useful for fully processed seismic data or for creating synthetic seismograms to aid in the interpretation of such data. A zero-phase wavelet is one whose phase is precisely zero, and it can be shown to be symmetric about $t = 0$. Such a symmetry means the wavelet is noncausal and hence cannot be directly produced by a physical source. The best a real source can do is a time-delayed symmetric wavelet, which would have linear, not zero, phase. In contrast, a minimum-phase wavelet has a phase spectrum that must be computed from its amplitude spectrum. Minimum-phase wavelets are useful in seismic modeling intended to simulate raw data. In data processing, any filter applied before deconvolution should be minimum phase, while, after deconvolution, zero phase is more appropriate.

Perhaps the most intuitive way to filter data is in the frequency domain. After a forward Fourier transform, the filter is designed and applied to the data by multiplication. The filtered time-domain data is recovered by an inverse Fourier transform. Letting $\hat{w}(f)_\mathrm{b}$, $\hat{w}(f)_\mathrm{h}$, and $\hat{w}(f)_\mathrm{l}$ stand for the spectra of band-pass, high-pass, and low-pass filters,

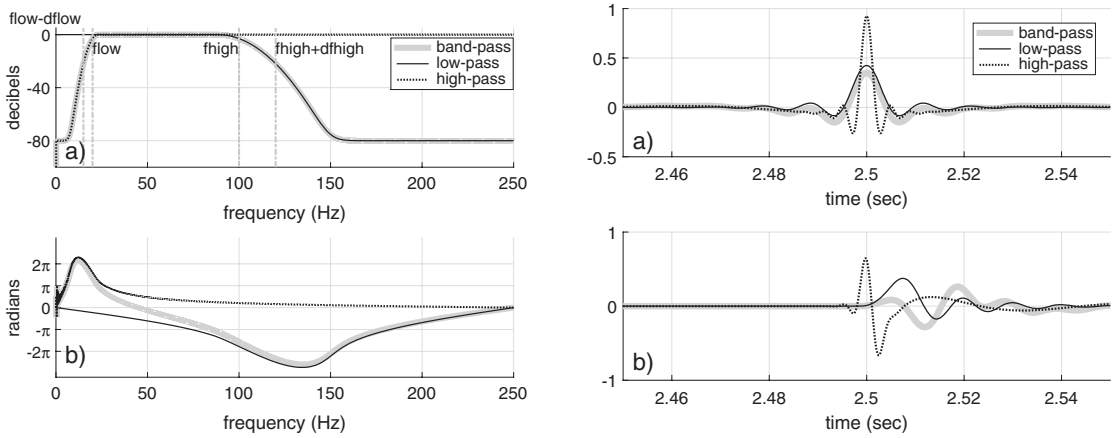


Figure 3.18a (left) Examples of the three common filter types are shown in the frequency domain. (a) The amplitude spectra of low-pass, high-pass, and band-pass filters. The Nyquist frequency is 250 Hz. (b) The minimum-phase spectra corresponding to the amplitude spectra in panel a. The zero-phase spectra are simply zero and are not shown. The phase spectra have been unwrapped.

Figure 3.18b (right) Impulse responses of the filters defined by the spectra in Figure 3.18a for (a) zero phase and (b) minimum phase. In both cases the impulse responses were computed at $\Delta t = 0.002$ s and resampled with an 8-point sinc interpolator to $\Delta t = 0.0002$ s for a smooth display. The impulse was placed at $t = 2.500$ s. The noncausal interpolator has created a slight precursor in the minimum-phase high-pass impulse response.

and letting $\hat{s}(f)$ be the Fourier-transformed data, then the corresponding filtered data is

$$\begin{aligned} s_{\mathrm{b}}(t) &= F^{-1}\left[\hat{s}\hat{w}_{\mathrm{b}}\right](t), \\ s_{\mathrm{h}}(t) &= F^{-1}\left[\hat{s}\hat{w}_{\mathrm{h}}\right](t), \\ s_{\mathrm{l}}(t) &= F^{-1}\left[\hat{s}\hat{w}_{\mathrm{l}}\right](t), \end{aligned} \tag{3.79}$$

where $F^{-1}[.]$ is the inverse Fourier transform. The filter spectra for typical low-pass, high-pass, and band-pass filters are illustrated in Figure 3.18a. These were designed by *filtspec* and can be applied to data by calling *filtf*. The calling syntax for *filtspec* is

```
[fltr,f]=filtspec(dt,tmax,fmin,fmax,phase,max_atten);
```

Here `dt` = Δt is the time sample size, which determines the Nyquist frequency; `tmax` is the signal length, which determines Δf, the frequency sample size; and `fmin` and `fmax` are two-element vectors of the form `[flow, dflow]` and `[fhigh, dfhigh]` that determine the filter passband. A filter amplitude spectrum is commonly specified by giving the frequencies `flow` and `fhigh` at which the spectrum is 3 dB down on the low- and high-frequency ends. The filters shown in Figure 3.18a have a Gaussian-shaped rolloff that is 3 dB down from maximum at `flow` or `fhigh`, and the standard deviations of the Gaussians are `dflow` and `dfhigh`. A low-pass filter is flagged by setting `fmin=0`, while a high-pass

filter uses fmax=0 (it is not necessary to specify fmin or fmax as two-element vectors in this instance). The phase parameter in *filtspec* can be either 0 or 1, indicating zero phase or minimum phase, respectively. Figure 3.18a shows only the minimum-phase spectra, because the zero-phase spectra are trivially zero. Close examination shows that the phase of the band-pass filter is a combination of the phases of the low-pass and high-pass ones. The major features of the phase spectrum are associated with the filter rolloff portions of the amplitude spectrum. The max_atten argument, which defaults to −80 dB, is specified in decibels and defines the maximum attenuation of the filters. Thus, the three filters are created by

```
dt=.002;tmax=5;
flow=20;dflow=5;
fhigh=100;dfhigh=20;
fmin=[flow dflow];
fmax=[fhigh dfhigh];
[flowpass,f]=filtspec(dt,tmax,0,fmax,1);%Lowpass
fhighpass=filtspec(dt,tmax,fmin,0,1);%highpass
fbandpass=filtspec(dt,tmax,fmin,fmax,1);%bandpass.
```

Plotting, for example, flowpass uses abs(flowpass) and unwrap(angle(flowpass)) for the amplitude and phase spectra.

Normally, *filtspec* is not called directly; rather, *filtf* is called with the syntax

```
sf=filtf(s,t,fmin,fmax,phase,max_atten);
```

where fmax, fmin, phase, and max_atten are the same as for *filtspec*, while s and t specify the signal to be filtered. Figure 3.18b shows the result of filtering an impulse to produce the *impulse response* of each filter. The impulse response of a filter is often simply called "the filter," or sometimes the "wavelet," and can be convolved with any signal to filter that signal. Here the impulse was a time series sampled at $\Delta t = 0.002$ s that was all zeros except for a single 1.0 at $t = 2.5$ s (*impulse* can conveniently make such an impulse). In the display of these impulse responses, all were resampled to $\Delta t = 0.0002$ s (i.e., 10 times finer than the Δt at which they were calculated) using *interpbl*. The zero-phase impulse responses are all symmetric about the impulse location, while the minimum-phase responses are all one-sided to times greater than that of the impulse.

We should mention two other filtering functions: *butterband* and *filtorm*. The former uses MATLAB's commands *butter* and *filter* to design and apply a *Butterworth* filter. These commands are found in the *signal* toolbox, which must be purchased separately in order for *butterband* to work. *filtorm* designs an Ormsby wavelet and applies it by convolution. Butterworth filters are commonly used, especially in engineering, and are designed to be optimally smooth in the frequency domain and have minimal sidelobes (ripple) in the time domain. The details of the design of these filters (Butterworth and Ormsby) will not be presented, as they are readily available elsewhere. Rather, we will simply present an example of their use in comparison with *filtf*. Code Snippet 3.6.1 demonstrates the use of these three different filtering tools by applying them to an artificial reflectivity series. In order to view the impulse response of each filter, the reflectivity series is modified by placing an isolated large spike in the midst of a field of 100 zeros

Code Snippet 3.6.1 This is an example of filtering a reflectivity series using three different filters. The reflectivity series is designed to contain an isolated spike in the last 100 samples to show the impulse response of the filter. The filters are applied with *filtf*, *butterband*, and *filtorm*. The filter parameters are designed to be as similar as possible for the three filter functions. The intent is to apply a 10–60 Hz band-pass filter in both minimum-phase and zero-phase modes. The basic specification is done on line 6 for *filtf* and *filtorm*, while the additional line 7 is needed for *butterband*. The results are shown in Figures 3.19a and 3.19b.

```
1   dt=.002;%time sample size
2   tmax=1.022;%Picked to make the length a power of 2
3   [r,t]=reflec(tmax,dt,.2,3,4);%reflectivity
4   r(end-100:end)=0;%zero the last 100 samples
5   r(end-50)=.1;%put a spike in the middle of the zeros
6   fmin=[10 5];fmax=[60 20];%used for filtf
7   n=4;%Butterworth order
8   sfm=filtf(r,t,fmin,fmax,1);%minimum phase filtf
9   sfz=filtf(r,t,fmin,fmax,0);%zero phase filtf
10  sbm=butterband(r,t,fmin(1),fmax(1),2*n,1);%min phase butterworth
11  sbz=butterband(r,t,fmin(1),fmax(1),n,0);%zero phase butterworth
12  som=filtorm(r,t,fmin(1)-fmin(2),fmin(1),fmax(1),fmax(1)+fmax(2),1);
13  soz=filtorm(r,t,fmin(1)-fmin(2),fmin(1),fmax(1),fmax(1)+fmax(2),0);
```

End Code

signalcode / filtering .m

at the end of the series. This spike is then replaced with the filter impulse response when each filter is applied. The example demonstrates a 10–60 Hz band-pass filter and attempts to choose parameters for each filtering tool to get the most similar results possible. For *filf*, the band-pass is specified by `fmin=[10 5]` and `fmax=[60 20]`. This defines the basic 10–60 Hz passband with a 5 Hz wide rolloff on the low end and a 20 Hz wide rolloff on the high end. A similar Ormsby filter would use `f1=fmin(1)-fmin(2)`, `f2=fmin(1)` to specify the low end and `f3=fmax(1)`, `f4=fmax(1)+fmax(2)` for the high end. The Butterworth filter specifies the rolloff differently, using an "order" parameter rather than an explicit width. The order controls the rapidity of the rolloff and is usually an integer in the range 2–10. It is a relatively simple matter to iterate and find an acceptable order by examining a few test results. It is a quirk of the Butterworth design that the filter is naturally minimum phase. To achieve a zero-phase result, the minimum-phase filter is applied twice, with the second pass using a time-reversed filter. This means that the "order" specification is doubled for the zero-phase result. Therefore, to get a similar amplitude spectrum in each case, Code Snippet 3.6.1 uses an order of 8 in the minimum-phase case and 4 in the zero-phase case. The results of minimum-phase filtering are shown in Figure 3.19a and for the zero-phase case in Figure 3.19b. Inspection of these figures suggests that all three methods give very similar results in the zero-phase case, while in the minimum-phase case the Butterworth impulse response appears to have considerably more phase rotation than the other two. Another apparent difference is that the Butterworth spectra appear to be

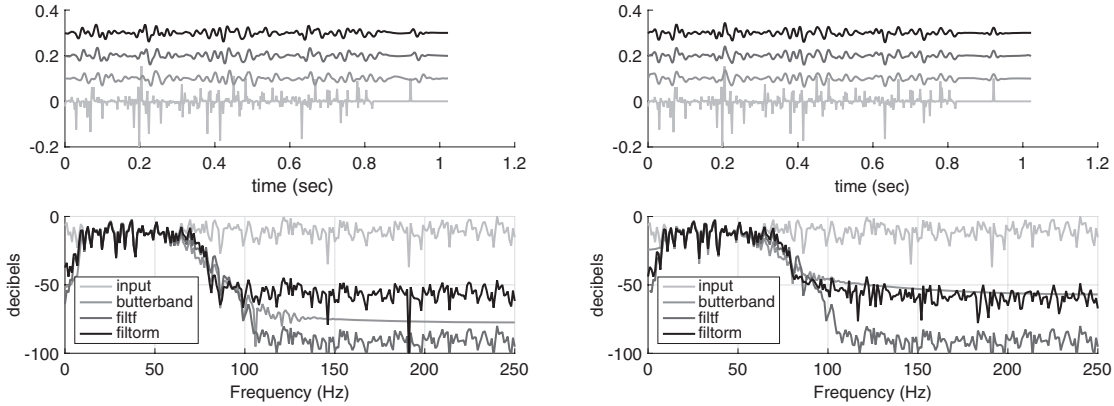


Figure 3.19a (left) Produced by Code Snippet 3.6.1, a reflectivity series is shown after 10–60 Hz minimum-phase band-pass filtering by three different filtering functions. An isolated spike placed at the end of the reflectivity allows inspection of the impulse response.

Figure 3.19b (right) Similar to Figure 3.19a except that the band-pass filter was applied as zero phase.

smooth and featureless at higher frequencies when compared with the other two. In the next section we will show that this is an artifact caused by truncation rather than tapering of the signal.

A common application of band-pass filters is the creation of filter panels that allow an assessment of the signal-dominated portion of the spectrum. Such panels also facilitate the understanding of the differing frequency content of various seismic wave types. For example, Rayleigh waves (or ground roll) typically are dominated by low frequencies, while body waves (or reflection data) show a much higher frequency content. Filter panels also allow an intelligent determination of the frequencies where noise overwhelms signal. The basic idea is to define a set of overlapping frequency bands that sample a plausible frequency range and then to display these in a way that allows their easy comparison. Code Snippet 3.6.2 shows a simplified generation of filter panels for the shot record contained in smallshot.mat. Since this is a raw shot record, an initial gain correction is performed (line 5) to approximately remove the effects of spherical spreading. Then five filter panels are specified (lines 7 and 8), where the first panel is essentially a broadband view and the other four are overlapping slices designed to examine 0–60 Hz. The filter panels are then computed in a loop using `filtf`. It is important to use a band-pass filter method capable of large attenuation because raw seismic data has a very large dynamic range. For example, if the 40–60 Hz band is 60 dB down from the 0–10 Hz band, then a 40–60 Hz band-pass filter capable of no more than 60 dB rejection will only equalize the two bands rather than showcase the 40–60 Hz band. `filtf` is a good choice for filter panels because the attenuation can be specified. Also computed in the loop is the average amplitude spectrum of each filter slice, obtained by calling `aveampspectrum`. The results of this code are displayed in Figures 3.20a and 3.20b. Examination of the filter panels shows that the Rayleigh waves, which are the steeply dipping events labeled A on the 0–10 Hz panel, are indeed restricted

Code Snippet 3.6.2 Here the shot record contained in *smallshot.mat* is separated into filter panels. Each filter panel is created by applying a certain zero-phase band-pass filter to the shot record. Prior to filtering, the shot record is gained using *gainmute* (line 5) to correct approximately for spherical divergence. The band-pass parameters of the filter panels are specified in cell arrays on lines 7 and 8. The first panel essentially does nothing and is the broadband record. Then band-pass filters at 0–10 Hz, 10–20 Hz, 20–40 Hz, and 40–60 Hz are defined to create five panels in total. The panels themselves are calculated in a loop and displayed using *imagesc* in Figure 3.20a. As each panel is calculated, the average amplitude spectrum for the panel is calculated by *aveampspectrum*. These are displayed in Figure 3.20b.

```
1   load ..\data\smallshot
2   % select space-time window and apply simple apply gain
3   ind=near(t,0,1.5);%time window
4   indx=near(x,0,950);%space window
5   seisg=gainmute(seis(ind,indx),t(ind),x(indx),max(x),...
6       [0 max(x(indx))],[0 0],1);
7   %define filter panels using cell arrays
8   fmins={0, 0, [10 3], [20 3], [40 5]};%cell array of fmin specs
9   fmaxs={[230 10], [10 5], [20 5], [40 5], [60 5]};%fmax specs
10  seisf=cell(size(fmins));%cell array for the panels
11  As=seisf;%cell array for average amplitude spectra
12  figure
13  for k=1:length(seisf)
14      subplot(1,5,k)
15      seisf{k}=filtf(seisg,t(ind),fmins{k},fmaxs{k},0,80);%filter
16      imagesc(x(indx),t(ind),seisf{k});%image plot in current axes
17      [As{k},f]=aveampspectrum(seisf{k},t(ind));%ave amp spectra
18  end
19  colormap('seisclrs')%install the seisclrs colormap
```

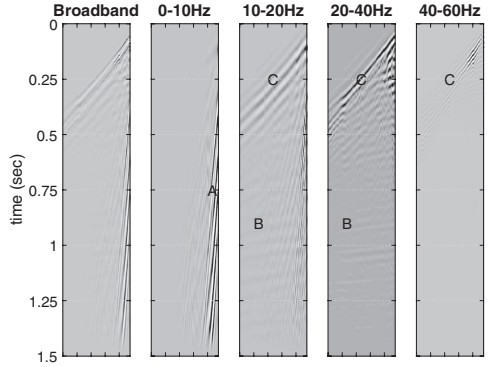
End Code

signalcode / filterpanels .m

to the low frequencies. Reflections, visible as the hyperbolic events labeled B, are seen in the 10–20 Hz and 20–40 Hz bands. First breaks, labeled C, are seen in all bands except 0–10 Hz. This figure is meant as a simplified example; in practice, more filter slices that examine a broader frequency range with perhaps more narrow passbands are common. Figure 3.20b shows the average amplitude spectra of each filter panel. Within the passband of a given filter panel, the spectrum matches the broadband spectrum, while outside that passband it decays sharply. The attenuation limit on these filters was 80 dB, and it is noteworthy that the peak amplitude at about 7 Hz in the broadband spectrum can be seen on the 40–60 Hz filter spectrum but reduced by 80 dB.

3.6.2 Spectral Analysis

The concept of spectral analysis is closely related to, but distinct from, that of filter panels. In both cases we are interested in learning more about the frequency content of data.



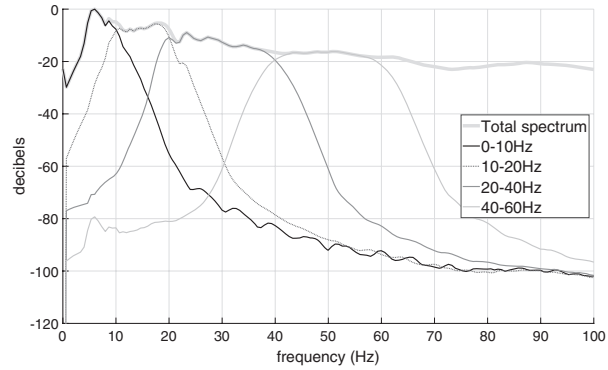


Figure 3.20a (left) The shot record contained in the dataset *smallshot.mat* (see also Figure 1.12a) is shown separated into filter panels. Each panel was created by applying a band-pass filter with $filtf$ as shown in Code Snippet 3.6.2. The filter passbands were designed to overlap slightly and were chosen based on experience. Letters denote wave types: A = Rayleigh waves, B = reflections, C = first breaks.

Figure 3.20b (right) The average amplitude spectra of each filter panel of Figure 3.20a. The computation is detailed in Code Snippet 3.6.2.

However, with the use of filter panels, the goal is to compare the character of spatially adjacent signals in sets of frequency bands and so make judgments about signal and noise content. In spectral analysis, the interest is more in the overall shape of the spectrum of a signal than in trying to decide what is signal and what is noise. In fact, it is extremely difficult to distinguish signal from noise using a single 1D spectrum. With filter panels, signal is judged to be present where there is spatial coherence between adjacent traces. Here we are more interested in obtaining an accurate representation of a signal's spectrum, and avoid any attempt to distinguish signal from noise. A major use of spectral analysis is in estimating the amplitude spectrum of the seismic wavelet (often referred to as the *embedded wavelet*).

Spectral analysis sounds deceptively easy. All we need to do is compute the DFT and and we are done. However, the goal is usually to deduce the spectrum of only a portion of the signal, and that makes things much more difficult. Consider the convolutional model $s(t) = (w \bullet r)(t)$ of a seismic trace. While this model is usually an oversimplification, it is sufficiently realistic to describe some of the difficulties. The exploration interest lies in the reflectivity $r(t)$ and not in $s(t)$. This means that, if somehow we could know $w(t)$, we could attempt to design an inverse filter, called a deconvolution operator, such that $r(t) = (d \bullet s)(t)$. So, in this context, the goal is to estimate the spectrum of $w(t)$, which is only a component of the observation $s(t)$. Another complication is that the seismic wavelet is known to evolve with traveltime, which means that, at best, the convolutional model is applicable only in some approximate, local sense. This complicates spectral analysis because it means that wavelet spectral estimates are best done on short segments of traces rather than the entire trace. A second consideration is that $r(t)$ is generally considered to

be a complicated, almost chaotic time series and hence its spectrum should be similar to that of white noise. As its name suggests, white noise has a nearly constant power at all frequencies but with chaotic, random fluctuations about this constant power level. The seismic wavelet is thought to be temporally short and relatively smooth. Therefore, in spectral analysis for the estimation of the wavelet (also called the source signature), the interest is in obtaining a smooth spectral estimate assuming that the spectral detail is from the reflectivity. So, we want a spectral analysis method that works on short portions of signal and produces smooth spectral estimates.

Selecting a portion of a signal for spectral analysis is known as windowing. This involves multiplying the signal by a second function, called the *window* or sometimes a *bump function*. Let $\Omega(t)$ denote a window function which has the basic property that its maximum is at $t = 0$, and far from the origin it becomes very small. Most window functions also are never negative. Some typical windows are the *boxcar*, the *triangle*, and the *Gaussian*, as shown in Figure 3.21a. These windows were created with the commands `boxkar`,[11] `triangle`, and `gaussian`. The widths of the windows in this figure have been adjusted to have roughly equal weight, which means that the triangle and Gaussian are about twice as wide as the boxcar. Then a windowed trace is $s_k(t) = s(t)\Omega_k(t)$, where $\Omega_k(t) = \Omega(t - t_k)$ denotes the time-shifted window centered at time $t = t_k$. Then, the convolution theorem states that

$$\hat{s}_k(f) = \left(\hat{s} \bullet \hat{\Omega}_k\right)(f), \tag{3.80}$$

so the spectrum of the windowed trace is the convolution of the trace spectrum with the window spectrum. Since the identity operation under convolution is to convolve $\hat{s}(f)$ with $\delta(f)$, then the only window which does not distort $\hat{s}(f)$ is the inverse Fourier transform of $\delta(f)$, and that is no window at all. Examination of the spectra in Figure 3.21a shows that the boxcar has the least favorable properties compared with the triangle and the Gaussian. The reason for this is the discontinuity of the boxcar at its endpoints, while the triangle is continuous with a discontinuous first derivative and the Gaussian is continuous for all derivatives. This causes the spectrum of the boxcar to decay much more slowly with increasing frequency than those of the other windows. Figure 3.21b shows that modifying the boxcar by tapering its edges improves its performance. Even a short taper over 10% of the total boxcar width has a marked effect. The taper used here is an option within `boxkar` and is called a *raised cosine*, given by $(1 + \cos(\pi(t - t_1)/(t_2 - t_1)))/2$, where t_1 is the time of the start of the taper and t_2 is the time of the end. (This formula is for the right-hand side of the boxcar, and a slightly modified one is used for the left-hand side.) The advantage of the boxcar and the triangle over the Gaussian is their *compact support*. This is a mathematical term meaning that these windows are nonzero only in a short interval. A signal windowed by a boxcar can be stored in a much smaller array than one windowed by a Gaussian, and this can be a significant advantage for large datasets. The Gaussian can be truncated at about four standard deviations, as shown in Figure 3.21a, with good spectral performance; however, this is twice the width of the triangle window.

[11] The name "boxcar" is already taken by a function in the `signal` toolbox.

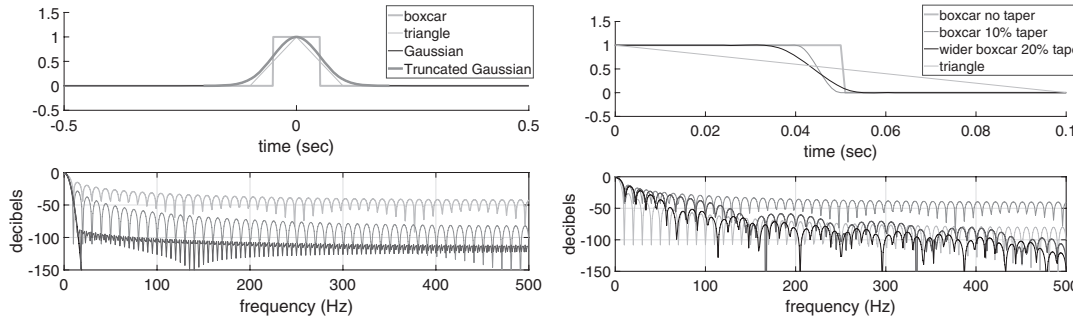


Figure 3.21a (left) Three common window functions are shown, the boxcar, triangle, and Gaussian, in the time domain (top) and the frequency domain (bottom). The window parameters have been chosen to given similar widths.

Figure 3.21b (right) The effect of tapering the boxcar window of Figure 3.21a is illustrated. The time-domain picture (top) is shown enlarged near the right-hand discontinuity of the boxcar. The untapered boxcar and the triangle window are the same as in Figure 3.21a. Also shown are a boxcar of the same width with a 10% taper and a slightly wider boxcar with a 20% taper.

In Figure 3.22a are shown the results from computation of windowed spectra on a convolutional synthetic seismogram. The seismogram was formed in the standard way as $s(t) = (w \bullet r)(t)$ using a synthetic reflectivity computed from *reflec* and a minimum-phase wavelet from *wavemin*. This is a stationary synthetic, which means that the embedded wavelet should be the same in any time interval. In this example, spectral analysis is attempted in a short window placed at the center of the seismogram ($t = 0.5$ s). For the boxcar window, the analysis consisted of

```
box=boxkar(t,tmax/2,tmax/10,1,0);
sb=s.*box;
```

where `tmax` is the maximum time of the signal, `s` and `t` are the convolutional seismogram and its time coordinate, and `sb` is the windowed signal. Notice that the window is applied with the `.*` operator. This window application is repeated for the other windows and the result is passed to *dbspec*, which is a utility program that computes and displays amplitude spectra. If `sg`, `st`, `w` are the Gaussian-windowed and triangle-windowed signal and the wavelet, then the amplitude spectra are computed and plotted by

```
dbspec(t,[s st sg w],'graylevels',[.8 .3 .3 0],'linewidths',...
    '[.5,.5,.5,.5],normoption',1,'linestyles','-',':','-','-');
```

The first two inputs to *dbspec* are the time coordinate vector and a matrix formed by the concatenation of four identically sized column vectors whose spectra are desired. The remaining arguments are all optional and consist of name/value pairs that customize the display. In this case the display is being changed from the default color plot to a gray-level display. The interested reader should consult the online documentation for *dbspec*.

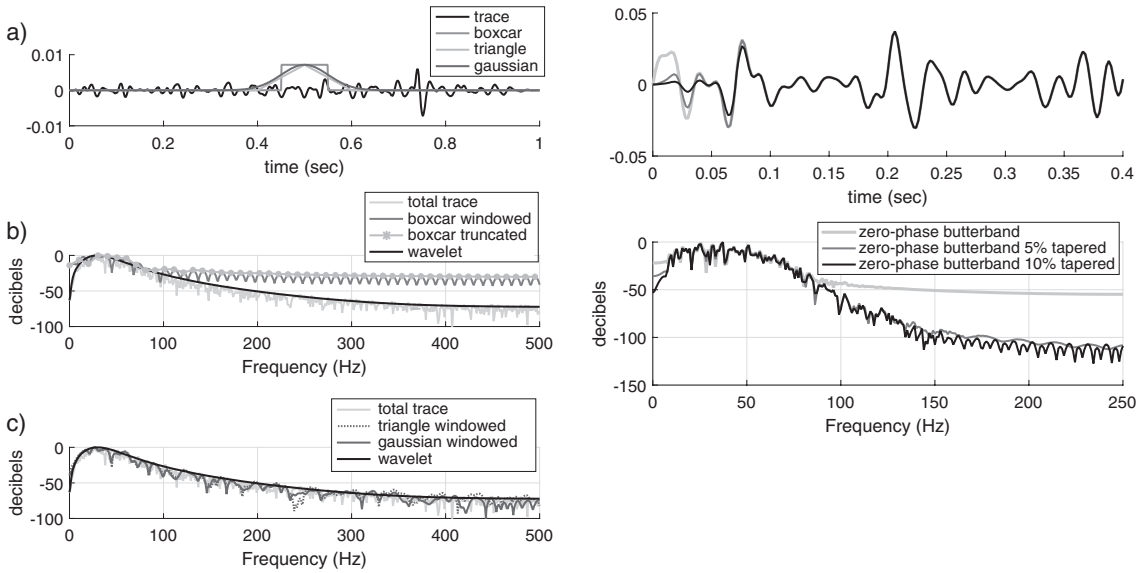


Figure 3.22a (left) (a) A synthetic seismogram created by $s(t) = (w \bullet r)(t)$ is depicted along with three spectral-analysis windows. (b) A comparison between the total spectrum of s and that estimated after boxcar windowing and after boxcar windowing plus truncation. The wavelet spectrum is also shown. (c) Similar to panel b except that the spectra estimated after triangle windowing and Gaussian windowing are shown.

Figure 3.22b (right) The zero-phase Butterworth result shown in Figure 3.19b is reexamined first by repeating the previous result and then by applying a 5% and a 10% raised-cosine taper to both ends of the signal. The taper is applied by generating an mwindow (using `mwindow`) and applying it to the `butterband` result with the `.*` operator. The featureless smooth spectrum above 100 Hz in the previous result is shown to be a truncation artifact.

In Figure 3.22a, a boxcar window with width 0.1 s is shown along with triangle and Gaussian windows of width 0.2 s. Here "width" refers to the width parameter in the functions `boxkar`, `triangle`, and `gaussian`. For the boxcar and triangle, the width is obviously the size of their compact support, while for the Gaussian it is four standard deviations. A copy of the embedded wavelet can be seen at about $t = 0.75$ s and it is somewhat shorter in length than the boxcar window. The performance of the boxcar window is revealed in the middle panel, which shows the spectral estimate resulting from the boxcar along with the seismogram spectrum (unwindowed) and the wavelet spectrum. As the goal here is to somehow estimate the wavelet spectrum, it is apparent that the windowed spectra are very poor estimates above about 90 Hz. This has resulted because of the relatively slow decay of the spectrum of the boxcar, as shown in Figure 3.21a. When this spectrum is convolved with the spectrum of the seismogram, as in Eq. (3.80), the result is this flattened spectrum above 90 Hz. Also shown is the spectrum that results from truncating the boxcar-windowed signal so that only the nonzero portion of $s(t)\Omega_k(t)$ is transformed. This truncated signal is much shorter than the original windowed signal but all of the truncated samples are simply zero. Since the frequency sample size is the inverse of the signal

length, the truncated signal has a much larger Δf and hence fewer frequency-domain samples. Since both the windowed and the windowed-truncated signals have the same Δt, the truncated signal must span $f = 0 \rightarrow f_{\text{nyq}}$ with far fewer samples than the windowed signal. Yet both spectra must display the same information, because the only thing different about them is a lot of zeros. This effect is seen in Figure 3.22a (central panel) in that the spectrum of the windowed-truncated signal follows the peaks of the spectrum of the windowed signal. In the lower panel of the same figure, the superior performance of the triangle and Gaussian windows is apparent. Both produce spectral estimates that follow the wavelet spectrum reasonably well for the entire frequency band.

As a final example, it was mentioned in the previous section that the loss of features in the Butterworth spectra above 100 Hz in Figures 3.19a and 3.19b is a truncation artifact. As the effect is more pronounced in the second figure, we will concentrate on the zero-phase case. Close inspection of Figure 3.19b indicates that the *butterband* result has larger amplitudes for $0 < t < 0.1$ than the other results have. Also, the reflectivity is very weak near $t = 0$, so these larger amplitudes seem anomalous. The filter was applied with MATLAB's *filter* function and it seems likely that there is some sort of undesired end effect. From a sampling perspective, these large amplitudes at the beginning of the trace cause a problem because they do not transition nicely into the low amplitudes at the end of the signal. This means that the first time-domain alias to the left has an abrupt amplitude change from the amplitudes visible near $t = 0$. Without trying to debug *filter* to ascertain the root cause of this problem, we can simply apply a raised-cosine taper to reduce the problem. In fact, such tapers are commonly used wherever truncation is called for, such as in *convm* and *convz*. Figure 3.22b repeats the *butterband* result of Figure 3.19b and then includes two traces that are 5% and 10% tapered versions of the first signal. The taper is easily applied to both ends of the signal using *mwindow*[12] to generate a raised-cosine tapered boxcar window that is then applied with the `.*` operator. The spectra of these three signals are displayed in the lower half of the figure and it is apparent that better spectral estimates are obtained from the tapered results. It is quite generally true that truncation of a strong signal generates high-frequency noise that can hide weaker signals. A taper, even a short one, almost always helps.

3.7 Time–Frequency Analysis

Time–frequency analysis is an extension of spectral analysis, where the frequency content of a signal is monitored in short time slices so that changes in this content can be observed as the signal evolves over time. This type of analysis is particularly useful for nonstationary signals, where the characteristics of a signal can change over time. In the case of seismic data, anelastic effects in the transmission of the seismic wave cause a frequency-dependent attenuation of the energy in a traveling wavelet. Such a change in the frequency content over time becomes apparent under time–frequency analysis. This method can also

[12] *boxkar* could also be used.

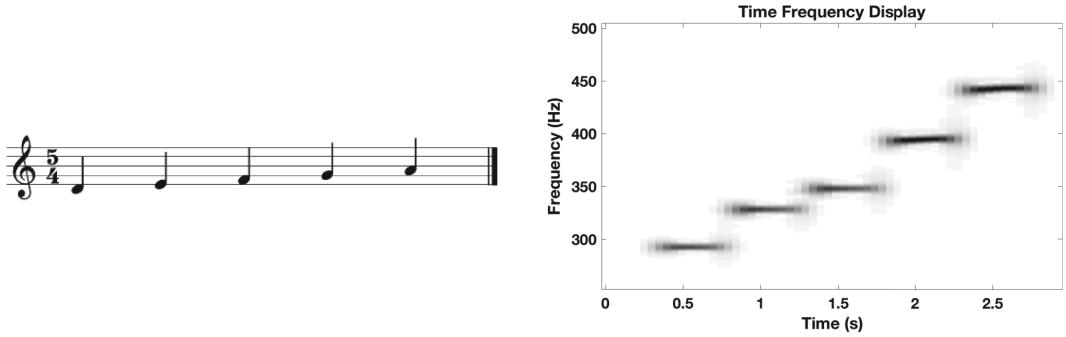


Figure 3.23a (left) A five-tone musical scale, as represented on the standard musical staff. These form the first five notes of a D minor scale, above middle C.

Figure 3.23b (right) A time–frequency display of the recorded musical scale. Note that the placement of the tones vertically indicates frequency, while the horizontal position indicates the time of activation for each note.

be used to introduce nonstationary filtering of signals, allowing for filters whose frequency response also changes over time.

Such an analysis begins with a time–frequency representation of a signal, which allows for a visual representation of the signal simultaneously in the time and frequency domains. A simplified model for a time–frequency representation is the musical notation used by all musicians, as represented in Figure 3.23a. The arrangement of dots on bars represent the various notes as they progress in time; in this example we see five notes in succession, with rising pitches.[13] A Fourier transform will identify the frequencies present in this progression, but not their placement in time. In contrast, the time–frequency display shown in Figure 3.23b clearly identifies both the frequencies of the notes (vertical axis) and their placement in time (horizontal axis). Indeed, the five ascending patches in the time–frequency display align very nicely with the symbolic representation of the notes shown in Figure 3.23a.

3.7.1 Gabor Transform

A specific type of time–frequency representation is given by the Gabor transform, which begins with a collection of window functions $\Omega_k(t)$, each centered at time t_k. These may be chosen in a variety of ways; for instance, in Dennis Gabor's original work, the window functions were set to be translates by t_k of a basic Gaussian of width w, defined as

$$\Omega(t) = e^{-t^2/w^2}. \tag{3.81}$$

[13] The first five notes of a D minor scale, above middle C.

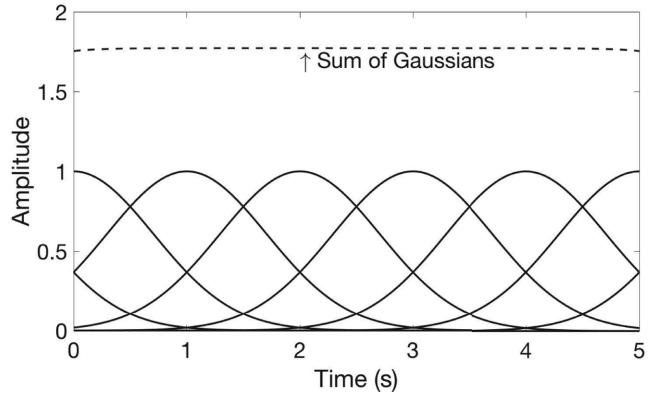


Figure 3.24 The sum of translated Gaussians is very nearly a constant function (the dotted line).

To obtain an invertible transform, we insist that this collection of windows sums to the constant function 1, which is to say

$$\sum_k \Omega_k(t) = 1 \text{ for all } t.$$

This is an example of a *partition of unity*. For instance, in the case of Gaussians, the uniformly translated Gaussians sum to nearly a constant function, as shown in Figure 3.24, so normalizing by dividing through by this sum gives the appropriate partition. Fixing a parameter p between 0 and 1, the corresponding analysis and synthesis windows are defined as

$$g_k(t) = \Omega_k^p(t), \qquad \gamma_k(t) = \Omega_k^{1-p}(t), \tag{3.82}$$

respectively.

The forward Gabor transform of a signal $s(t)$ is then defined as the Fourier transform of the signal windowed by each g_k, giving a function of two variables time t_k and frequency f as

$$\hat{s}_g(t_k,f) = G[s](t_k,f) = \int_{-\infty}^{\infty} s(\tau) g_k(\tau) e^{-2\pi i f\tau}\, d\tau. \tag{3.83}$$

The inverse Gabor transform is obtained by taking the inverse Fourier transform of each k-slice $\hat{s}_g(t_k,f)$ and then summing over the synthesis windows, yielding

$$s(t) = G^{-1}[\hat{s}_g] = \sum_k \gamma_k(t) \int_{-\infty}^{\infty} \hat{s}_g(t_k,f) e^{2\pi i f t}\, df. \tag{3.84}$$

It is easy to verify that this really does define a (left) inverse by using the partition-of-unity property, which states that $\sum_k g_k(t)\gamma_k(t) = \sum_k \Omega_k(t) = 1$. When the signal is band limited, the Fourier transform can be replaced by a discrete transform using the signal

Code Snippet 3.7.1 A simple chirp, or time-varying sinusoid, is created, and its Gabor transform computed. We use Gaussian windows of length "wlen" and a time increment of half the window length. The amplitude spectrum is computed on each windowed selection. We skip the phase correction for time-stepping, since only absolute values are displayed. The result is shown in Figure 3.25.

```
% One second chirp at 8000 sample rate
Fs = 8000;
t = linspace(0,1,Fs);
x = sin(2*pi*2000*t.^2);
% Gabor window, length and step size
wlen = 128;
wstep = wlen/2;
win = exp(-linspace(-1,1,wlen).^2);
% Gabor transform, 101 windows
gg = zeros(wlen,100);
for k=1:100
    gg(:,k) = abs(fft(x( k*wstep + (1:wlen) ).*win));
end
imagesc([0,1],[0,-Fs],-abs(gg)), colormap(gray)
```

End Code

signalcode / GaborChirp.m

samples $s_n = s(n\,\Delta t), g_{k,n} = g_k(n\,\Delta t)$ to obtain

$$\hat{s}_g(t_k,f) = G[s](t_k,f) = \Delta t \sum_n s_n g_{k,n} e^{-2\pi i f n\,\Delta t}. \tag{3.85}$$

In a numerical implementation of the Gabor transform, it is useful to take advantage of the fast Fourier transform. The time sample size Δt is chosen to be short enough to capture the frequency content of the signal $s(t)$ being sampled.[14] Next, the window functions $g_k(t)$ are chosen so that the samples $g_k(n\,\Delta t)$ are supported on some N sample points. The frequency sampling interval Δf is chosen to meet the Nyquist criterion $\Delta f\,\Delta t = 1/N$. The sampled Gabor transform is then expressed as the sum

$$G[s](t_k,f_j) = \sum_n s_n g_{k,n} e^{-2\pi i n j/N}, \tag{3.86}$$

where the t_k are the locations of the window functions g_k, and the $f_j = j\,\Delta f$ are frequency samples for the DFT. The sum is over the N points where $g_{k,n}$ is nonzero, so the DFT algorithm can be applied. Further efficiencies are obtained in the case where the window functions g_k are simply translated by t_k of some basic window g_0 centered at time $t = 0$.

A simple implementation of the Gabor transform is demonstrated in Code Snippet 3.7.1 using a collection of 101 equally spaced Gaussian windows. The transform is applied to

[14] Actually, Δt needs to be even shorter, to account for the bandwidth-broadening effects of multiplying by the window functions $g_k(t)$.

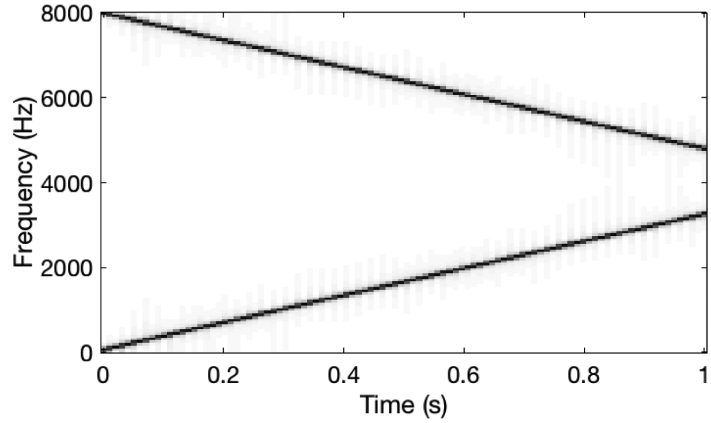


Figure 3.25 The Gabor transform of a chirp, showing both the rising tone and its mirror image in frequency.

a sinusoidal chirp signal of 1 s duration, with frequencies ranging from 0 to 4000 Hz. The resulting time–frequency display is shown in Figure 3.25.

3.7.2 Gabor Multipliers

A Gabor multiplier is a nonstationary filter obtained by modifying a signal in the Gabor domain. It is analogous to filtering in the Fourier domain by multiplying the Fourier transform $\hat{s}(f)$ of a signal by a frequency-dependent function $\alpha(f)$ to modify the frequency content of the signal. The extension to the Gabor domain allows the filter characteristics to change with time.

To construct a Gabor multiplier, or time-variant filter, a function $\alpha(t,f)$ is chosen that has the desired time–frequency characteristics. For instance, to filter out noise that is centered at some frequency f_0 near some time t_0, the function $\alpha(t,f)$ is chosen as

$$\alpha(t,f) = \begin{cases} 0 & \text{if } t,f \text{ near } t_0,f_0, \\ 1 & \text{if } t,f \text{ far from } t_0,f_0, \end{cases} \tag{3.87}$$

and we smoothly interpolate between these two regimes. A signal $s(t)$ is modified in the Gabor domain simply by multiplying $\hat{s}_g$ by the multiplier α to get the function

$$\alpha(t,f) \cdot \hat{s}_g(t,f).$$

The output of the time-variant filter is obtained by applying the inverse Gabor transform to this product. We can write this in the form

$$M_\alpha(s) = G^{-1}[\alpha \cdot \hat{s}_g]. \tag{3.88}$$

Note that $M_\alpha(s)$ is a signal in the time domain, as the Gabor inverse always returns a function of the single variable t.

A typical seismic application is to use a Gabor multiplier to model Q attenuation. This is a time- and frequency-dependent physical process that causes the seismic wave to lose energy as a function of both t and f. A simplified model would set α to represent an exponential decay, in the form

$$\alpha(t,f) = e^{-\pi |f| t/Q}. \tag{3.89}$$

3.7.3 Factoring a Time-Variant Reflectivity Model

We have seen that a model for the standard seismic experiment is to represent the seismic signal as a convolution of a wavelet $w(t)$ with the reflectivity $r(t)$ of the Earth, written as

$$s(t) = (w \bullet r)\,(t)\,. \tag{3.90}$$

In the Fourier domain, this factors as the product of transforms

$$\hat{s}(f) = \hat{w}(f)\hat{r}(f) = \hat{w}(f) \int_{-\infty}^{\infty} r(t) e^{-2\pi i f t}\, dt. \tag{3.91}$$

For a nonstationary model, as discussed in Section 5.6.1, a time-varying factor $\alpha(t,f)$ is inserted into the integral to obtain the general form

$$\hat{s}(f) = \hat{w}(f) \int_{-\infty}^{\infty} \alpha(t,f) r(t) e^{-2\pi i f t}\, dt. \tag{3.92}$$

The integral in Eq. (3.92) is an example of a pseudodifferential operator, and unfortunately we no longer have a factorization in the Fourier domain with such time-varying operators.

However, in the Gabor domain we do obtain an approximate factorization of Eq. (3.92), to find that

$$\hat{s}_g(t,f) \approx \hat{w}(f)\alpha(t,f)\hat{r}_g(t,f), \tag{3.93}$$

which says that the Gabor transform of the seismic signal s is approximately the product of the Fourier transform of the wavelet w, the attenuating function $\alpha(t,f)$, and the Gabor transform of the reflectivity r.

To derive this factorization, we need to assume that the attenuation function $\alpha(t,f)$ is slowly changing with respect to the partition of unity Ω_k, in which case the approximation

$$\alpha(t,f) \approx \sum_k \Omega_k(t)\alpha(t_k,f) \tag{3.94}$$

holds. Inserting this approximation into Eq. (3.92) yields

$$\hat{s}(f) \approx \hat{w}(f) \sum_k \alpha(t_k,f) \int_{-\infty}^{\infty} \Omega_k(t) r(t) e^{-2\pi i f t}\, dt = \hat{w}(f) \sum_k \alpha(t_k,f)\hat{r}_k(f), \tag{3.95}$$

where $r_k(t) = \Omega_k(t)r(t)$ is the localized reflectivity. This is a sum of products in the frequency domain, so returning to the time domain gives a sum of convolutions,

$$s(t) \approx \sum_k (w \bullet a_k \bullet r_k)(t), \tag{3.96}$$

where $a_k(t)$ is the inverse Fourier transform of the function $\alpha(t_k,f)$ and the convolution $w \bullet a_k$ is the propagating wavelet. The product with the jth analysis window g_j gives

$$g_j(t)s(t) \approx \sum_k g_j(t)(w \bullet a_k \bullet r_k)(t) \approx \sum_k (w \bullet a_k \bullet (g_j \cdot r_k))(t), \tag{3.97}$$

where we can move the factor g_j into the convolution because the window g_j is much longer in time than the propagating wavelet $w \bullet a_k$ and so the product and convolution approximately commute. The doubly windowed product $g_j \cdot r_k = g_j \cdot \Omega_k \cdot r$ is usually zero, except for when the window functions g_j, Ω_k overlap. This only happens when k,j are close, and since the multiplier function $\alpha(t_j,f)$ is slowly varying with respect to the windows, we can replace a_k with a_j in the previous equation to get the approximation

$$g_j(t)s(t) \approx \sum_k (w \bullet a_j \bullet (g_j \cdot \Omega_k \cdot r))(t). \tag{3.98}$$

Recalling that the Ω_k form a partition of unity, we can sum over the k to obtain

$$g_j(t)s(t) \approx (w \bullet a_j \bullet (g_j \cdot r))(t). \tag{3.99}$$

Taking Fourier transforms of these windowed functions gives the Gabor transform (noting the convolutions turn into products under Fourier transformation), giving the desired approximate factorization

$$\hat{s}_g(t_j,f) \approx \hat{w}(f)\alpha(t_j,f)\hat{r}_g(t_j,f). \tag{3.100}$$

3.8 Multidimensional Discrete Fourier Transforms

The continuous multidimensional Fourier transform given in Eq. (2.132) has a discrete analog. First, the signal $u(t,x)$ is sampled in both the t and the x coordinates, yielding real-valued samples

$$u_{jk} = u(j\,\Delta t, k\,\Delta x), \qquad \text{for integers } 0 \le j < M, 0 \le k < N, \tag{3.101}$$

where Δt is the sample spacing in the time coordinate t, Δx is the spacing in the space coordinate x, and we are assuming the signal $u(t,x)$ is only nonzero inside the rectangle $[0, M\,\Delta t] \times [0, N\,\Delta x]$. The Fourier space is sampled in the (f,k) space with a uniform spacing of $\Delta f = 1/\Delta t$ and $\Delta k = 1/\Delta x$ to obtain the discrete transform

$$\hat{u}_{\nu\kappa} = \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} u_{jk} e^{-2\pi i(j\nu/M + k\kappa/N)}, \tag{3.102}$$

for integers $0 \leq \nu < M, 0 \leq \kappa < N$. The inverse transform is given by a similar double sum, where we flip the sign on the complex exponential:

$$u_{jk} = \frac{1}{MN} \sum_{\nu=0}^{M-1} \sum_{\kappa=0}^{N-1} \hat{u}_{\nu\kappa} e^{2\pi i(j\nu/M + k\kappa/N)}. \tag{3.103}$$

As in the one-dimensional case, there are fast Fourier transforms for the two-dimensional transform defined here, as well as for arbitrary n-dimensional DFTs. MATLAB provides the function $fftn$ to implement the fast transform on any n-dimensional array of signal samples. It is useful to recall that in the case of band-limited signals, the multidimensional DFT defined here is simply a sampling of the multidimensional continuous Fourier transform, which can be reconstructed exactly using a product of sinc functions. So, in principle, there is no loss of information in going from the continuous to the discrete domain for such signals.

With seismic signals, this band limiting is not always possible to achieve, particularly in the spatial sampling. The next section considers this issue.

3.8.1 Spatial Aliasing

In Section 3.2, the aliasing of time signals in 1D was discussed and the sampling theorem was presented. It was shown that only a band-limited signal can be sampled without aliasing and that the time sample interval must satisfy $\Delta t < 1/(2f_{\max})$, where $f_{\max}$ is the maximum frequency present in the band-limited signal. Since most real-world signals have no obvious band limit, the digital sampling of time signals must be concerned about possible aliasing. This concern is met in practice by the use of analog antialias filters. These are analog low-pass filters that ensure approximate band limiting before digital sampling. This is possible because the time signal usually exists as an electrical current in a circuit before it is sampled. Given this widespread practice, temporal aliasing is rarely encountered in seismic data.

However, it is a different story with spatial sampling. Here the wavefield, which exists in the Earth as a continuous spatial phenomenon, is never recorded with spatial continuity. Instead, the field is always sampled at discrete locations wherever receivers are placed, and there is no spatial antialias filter. Therefore, there is always the likelihood that some seismic events are recorded with spatial aliasing. While a spatial antialias filter seems impossible,[15] the physics of wave propagation provides some help by linking the frequency signal band with the wavenumber signal band. This was discussed in Section 2.5.2, where it was shown that, in a layered medium where velocity depends upon depth, the maximum wavenumber is given in terms of the maximum frequency by $|k_{\max}| = f_{\max}/v(z)$.

Consider the possibility of sampling in receiver position along a single spatial axis, x, and time, t, without aliasing in either dimension when the slowest velocity, v_0, occurs just beneath the receivers. How large will such a dataset be? Assume the receivers will be spread over a distance L and the temporal samples over a record length T. Then, the time

[15] The use of receiver arrays provides some partial antialias protection but this is never fully effective.

sample interval must be no greater than $\Delta t = 1/(4f_{\max})$[16] and the spatial sample interval must be no greater than $\Delta x = 1/(2k_{\max}) = v_0/(2f_{\max})$. Then, there will be $N_x = L/\Delta x$ receivers and each will record a trace with $N_t = T/\Delta t$ samples.[17] Thus the total number of samples of a single 2D shot record will be

$$N_{\text{shot2D}} = N_x N_t = \frac{L}{v_0/(2f_{\max})} \frac{T}{1/(4f_{\max})} = \frac{8LTf_{\max}^2}{v_0}. \tag{3.104}$$

Choosing some modest parameters such as $L = 5000\,\text{m}$, $T = 3\,\text{s}$, $f_{\max} = 150\,\text{Hz}$, and $v_0 = 1000\,\text{m/s}$, we find $\Delta x \approx 3.33\,\text{m}$, $\Delta t \approx 0.0017\,\text{s}$, and $N_{\text{shot2D}} \approx 2.7 \times 10^6$ samples. Assuming 4 bytes per sample, we expect a dataset size of about 12 MB (MB = megabyte), which seems very reasonable. However, the Earth is 3D and we should deploy our receivers over an area of L^2, so we predict, for 3D sampling without aliasing,

$$N_{\text{shot3D}} = N_x N_y N_t = \left(\frac{L}{v_0/(2f_{\max})}\right)^2 \frac{T}{1/(4f_{\max})} = \frac{16L^2Tf_{\max}^3}{v_0^2}. \tag{3.105}$$

With the same parameters as before, we have $N_{\text{shot3D}} \approx 4.0 \times 10^9$ samples, with dataset sizes in the 16 GB (gigabyte) range. These equations are estimates for a single source record and a modern seismic dataset can have many thousands of source locations. Seismic sampling theory (e.g., Vermeer (1990)) suggests that, in the ideal case, sources and receivers must be treated reciprocally so that we must have a source location at or near each receiver location. Assuming all receivers are live for each source, then the total dataset size in 3D is estimated at

$$N_{\text{3D}} = (N_x N_y)^2 N_t = \left(\frac{L}{v_0/(2f_{\max})}\right)^4 \frac{T}{1/(4f_{\max})} = \frac{64L^4Tf_{\max}^5}{v_0^4}, \tag{3.106}$$

and for 2D

$$N_{\text{2D}} = N_x^2 N_t = \left(\frac{L}{v_0/(2f_{\max})}\right)^2 \frac{T}{1/(4f_{\max})} = \frac{16L^2Tf_{\max}^3}{v_0^2}. \tag{3.107}$$

Again using the same parameters as before, we find $N_{\text{3D}} \approx 9.1 \times 10^{15}$ samples, which is about 36 PB (1 PB = petabyte = 10^6 GB), which is very prohibitively large. As large as this is, a 5×5 km survey is actually small and so the possibility of sampling modern 3D datasets without any aliasing at all looks slim. (N_{2D} is the same as N_{shot3D}.) Clearly, some sampling compromises must be made and the likely presence of spatially aliased events in real datasets must be accepted. Those interested in seismic survey design, which is a complex task, should look elsewhere (e.g., Vermeer (2012)). Here we will simply describe spatial aliasing.

Code Snippet 3.8.1 illustrates the creation of an idealized shot record, designed to have no spatial or temporal aliasing. The tools *event_dip* and *event_hyp* are used to create two different linear events and a set of four reflection hyperbolas. The linear events are

[16] We allow for the temporal antialias filter to operate over the higher frequencies and choose a sample size such that $f_{\max} = 0.5f_{\text{nyq}}$.

[17] Technically, these expressions should both have a "+1," but we ignore that as trivial.

Code Snippet 3.8.1 A synthetic shot record is created by inserting events of different geometries and bandwidths in a matrix. The design is such that there is no spatial aliasing. Three types of events are created: (1) linear first breaks (lines 8–13), (2) linear low-velocity noise (lines 15–18), and (3) hyperbolic reflection events (lines 20–24). Each event type is first inserted into its own matrix (see lines 5 and 6) so that different band-pass filters can be applied (lines 26–31). After filtering, the events are combined (line 32). Finally, the *f–k* transform is computed on line 34.

```
1  %geometry
2  dt=.004;tmax=1.0;t=(0:dt:tmax)';%time coordinate
3  dx=7.5;xmax=1000;x=-xmax:dx:xmax;%x coordinate
4  %preallocate seismic matrices
5  seis=zeros(length(t),length(x));%for hyperbolic events
6  seisn=seis;seisfb=seis;%for first breaks and noise
7  %first breaks
8  vfbl=2000;vfbr=2500;%first break velocities to the left and right
9  afb=3;anoise=4;%amplitude of first breaks and noise
10 t1=0;t2=xmax/vfbr;%times at 0 and far offset for right first break
11 seisfb=event_dip(seisfb,t,x,[t1 t2],[0 xmax],afb);
12 t1=0;t2=xmax/vfbl;%times at 0 and far offset for left first break
13 seisfb=event_dip(seisfb,t,x,[t1 t2],[0 -xmax],afb);
14 %noise
15 vnoise=1000;%noise velocity
16 t1=0;t2=xmax/vnoise;%times at 0 and far offset for noise
17 seisn=event_dip(seisn,t,x,[t1 t2],[0 xmax],anoise);
18 seisn=event_dip(seisn,t,x,[t1 t2],[0 -xmax],anoise);
19 %reflectors
20 vstk=2500:500:4000;%hyperbolic velocities of reflections
21 t0=[.2 .35 .5 .6];a=[1 -1 -1.5 1.5];%zero offset times and amps
22 for k=1:length(t0)
23     seis=event_hyp(seis,t,x,t0(k),0,vstk(k),a(k));
24 end
25 %filters
26 flow=10;delflow=5;fmax=60;delfmax=20;%bandpass filter params
27 ffbmax=80;delffb=10;%first break filter
28 fnoisemax=30;delnoise=10;%noise filter
29 seisf=filtf(seis,t,[flow delflow],[fmax delfmax],1);%filter signal
30 seisnf=filtf(seisn,t,0,[fnoisemax delnoise],1);%lowpass filter noise
31 seisfbf=filtf(seisfb,t,[flow delflow],[ffbmax delffb],1);%filter FBs
32 seisf=seisf+seisnf+seisfbf;%combined section
33 %fk transform
34 [seisfk,f,kx]=fktran(seisf,t,x);
```

End Code

signalcode / makesyntheticshot .m

low-velocity noise, with a velocity of 1000 m/s, and first breaks which have different velocities to the left and right (2000 m/s to the left and 2500 m/s to the right).[18] The three event

[18] The distinct left and right first-break velocities are intended to illustrate a basic point about *f–k* spectra later.

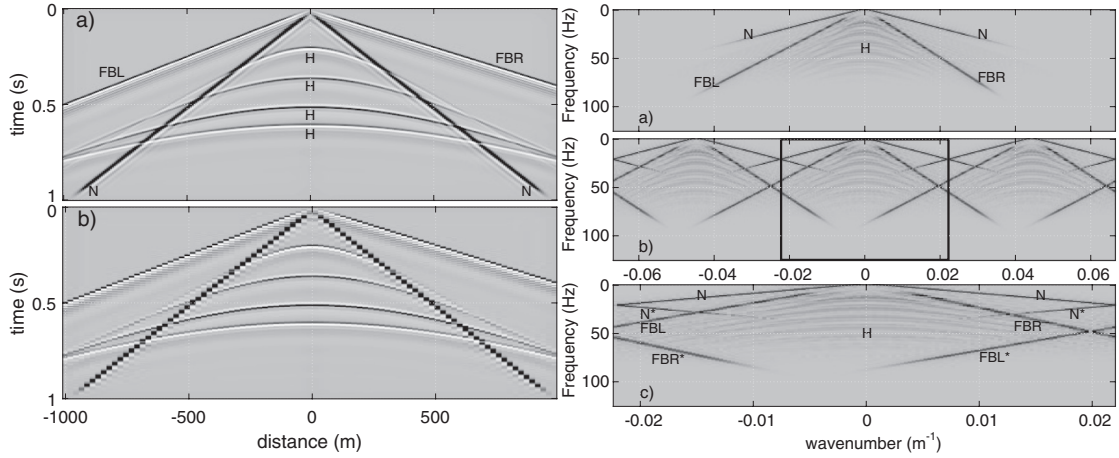


Figure 3.26a (left) (a) A synthetic shot record as created by Code Snippet 3.8.1. The labeled events are FBL = first break left, FBR = first break right, N = low-velocity noise, and H = hyperbolic reflection. The left and right first breaks have deliberately been given different velocities. The first breaks, noise, and reflections all have different bandwidths. The spatial sample interval is $\Delta x = 7.5$ m. (b) A version of the shot record in panel a formed by selecting every third trace to create a spatial sampling interval of $\Delta x = 22.5$ m. Spatial aliasing is present in this data.

Figure 3.26b (right) (a) The f–k amplitude spectrum of the shot record shown in Figure 3.26a. (a) The Nyquist wavenumber is $k_{\text{nyq}} = 0.5/\Delta x = 0.0667\ \text{m}^{-1}$ and there is no spatial aliasing. The event labels are the same as in Figure 3.26a, panel a. Note the differing bandwidths of the event types. (b) After spatial downsampling to $\Delta x = 22.5$ m, which has $k_{\text{nyq}} = 0.0222\ \text{m}^{-1}$, the spectral aliases to the left and right appear and overlap with the principal band (black box), creating spatial aliasing. (c) The principal band of panel b is shown enlarged. New events that are spatial aliases of the first breaks and the noise are indicated by "*" in their labels.

types are given different bandwidths, as this is often found in real data. The bandwidths are 0–30 Hz for the linear noise, 10–80 Hz for the first breaks, and 10–60 Hz for the reflections. This is accomplished by first inserting each event type in its own matrix, applying the filter, and then summing the three filtered matrices. The spatial sampling interval was $\Delta x = 7.5$ m, corresponding to a Nyquist wavenumber $k_{\text{nyq}} = 0.0667\ \text{m}^{-1}$, and the temporal sampling interval was $\Delta t = 0.004$ s, so $f_{\text{nyq}} = 125$ Hz. The result of this is shown in Figure 3.26a, panel a, and the f–k amplitude spectrum is shown in Figure 3.26b, panel a. The two families of linear events are easily discerned in both (t, x) space and (f, k_x) space, but notice that the first breaks are on the outside in (t, x) space and on the inside in (f, k_x) space. Notice also the different slopes of the first breaks on the left and right sides in both domains. The four individual hyperbolas are quite distinct in (t, x) space but their energy is spread over many apparent velocities in (f, k_x) space (compare with Figures 2.18a and 2.18b). The apparent hyperbolic shapes in (f, k_x) space are not related to individual events in (t, x) space but are spectral notches caused by interference between pairs of hyperbolic events.

Figure 3.26a, panel b, shows what happens after 3 : 1 spatial downsampling, which induces a lot of spatial aliasing. The downsampling was accomplished by simply selecting every third trace for the original unaliased shot to get a new spatial sampling interval

of $\Delta x = 22.5\,\mathrm{m}$ and $k_{\mathrm{nyq}} = 0.0222\,\mathrm{m}^{-1}$. Just as happened in time-domain sampling, space-domain downsampling causes the appearance of aliases of the principal spectrum to the left and right, as is shown in Figure 3.26b, panel b. This view of the spectrum[19] is not what is normally seen from a discrete *f–k* transform; instead, we normally just see the principal band, as shown in Figure 3.26b, panel c. The most obvious aliased events are those associated with the linear first breaks and noise and are identified by an asterisk after their labels in the figure. Comparing panels b and c allows an understanding that each aliased event is actually coming from a copy of the original spectrum either to the left or to the right. For example, the event labeled "FBR*" occurs on the left side of (f, k_x) space but is actually the alias of the first break on the right-hand side coming from the first alias to the left. The use of different velocities for the left and right first breaks was intended to make this more obvious. Similarly, the event on the right-hand side of panel c labeled "FBL*" has nothing to do the event "FBR" but instead comes from "FBL" on the first alias to the right. The hyperbolic events are not so obviously aliased, but they surely are.

For a linear event of known time dip, the critical frequency that marks the onset of spatial aliasing can easily be predicted. As shown in Eq. (2.142), such a linear event will be found on the *f–k* trajectory $f/k_x = v_{\mathrm{app}}$. The onset of spatial aliasing will occur at the frequency at which the event intersects $\pm k_{\mathrm{nyq}}$. This is given by

$$f_{\mathrm{crit}} = \left|v_{\mathrm{app}}\right| k_{\mathrm{nyq}} = \frac{\left|v_{\mathrm{app}}\right|}{2\,\Delta x}, \tag{3.108}$$

where Δx is the spatial sample size. If $f_{\mathrm{crit}} < f_{\mathrm{nyq}}$, then technically the event has spatial aliasing; however, the bandwidth of the event may be such that it has no significant energy at f_{crit}, in which case there is no significant spatial aliasing. For example, the linear noise in Figure 3.26a has an apparent velocity $v_{\mathrm{app}} = 1000\,\mathrm{m/s}$ and, for $\Delta x = 7.5\,\mathrm{m}$, then $f_{\mathrm{crit}} \approx 67\,\mathrm{Hz}$, while at $\Delta x = 22.5$, $f_{\mathrm{crit}} \approx 22\,\mathrm{Hz}$. Both of these critical frequencies are less than f_{nyq}, which is 125 Hz in this case. However, inspection of Figure 3.26a shows that the 0–30 Hz bandwidth of the event means that there is significant aliasing only at the larger Δx. For $f > f_{\mathrm{crit}} \approx 22\,\mathrm{Hz}$, the event aliases from one side of (f, k_x) space to the other. The time dip of any *f–k* point on the event is given by $dt/dx = v_{\mathrm{app}}^{-1} = k_x/f$, so when the event aliases and wraps around, the time dip changes sign. Also, each point on the aliased event will have a unique time dip because the aliased event is not a radial line from the origin. Figure 3.27 illustrates the aliasing of this event. A small ensemble of traces were selected from the left-dipping portion of the noise event created in line 30 of Code Snippet 3.8.1. In the upper panel of this figure, a set of six narrow band-pass filters are shown to isolate portions of the event in the vicinity of $f_{\mathrm{crit}} \approx 22\,\mathrm{Hz}$. The unaliased time dip of this event is down to the left and, in the first three filter panels, the alignment of traces in the ensemble agrees with this. However, in the second three panels, which are

[19] To create this view, rather than selecting every third trace from the original shot gather, the original gather was duplicated and then two out of every three traces were zeroed. This makes a shot record with the same information as is obtained by selecting every third trace, but the Nyquist wavenumber is still the original value, which permits the expanded view.

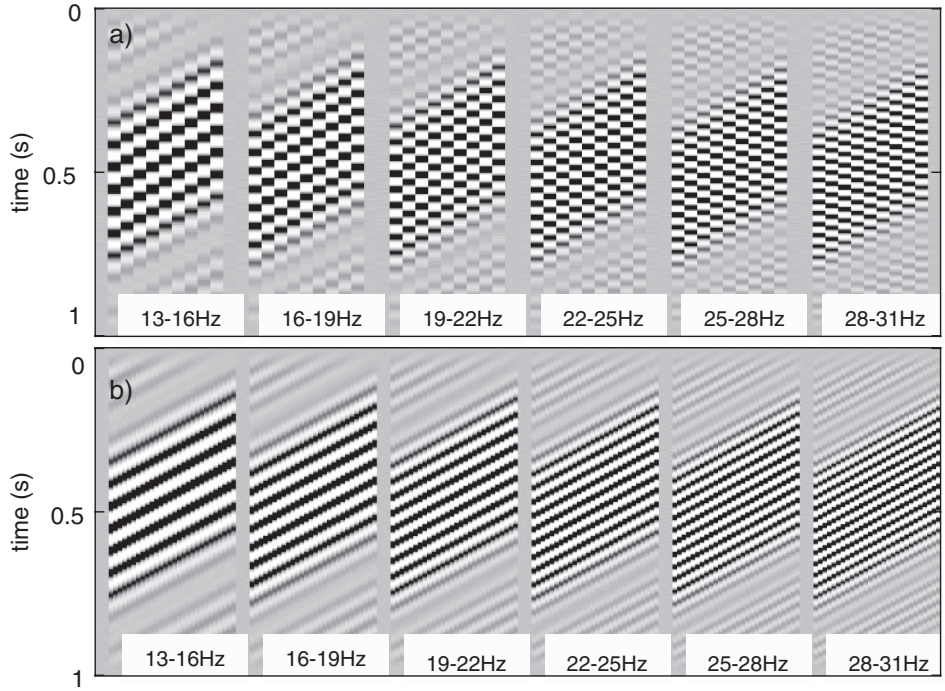


Figure 3.27 A portion of the noise event in Figure 3.26a is shown sliced into filter panels. (a) A nine-trace ensemble with $\Delta x = 22.5$ m sliced into five filter panels. The critical frequency of the onset of spatial aliasing for this event is $f_{\text{crit}} = 22.2$ Hz. (b) A 27-trace ensemble sampling the same event as in panel a but with $\Delta x = 7.5$ m. This event is not spatially aliased at these frequencies.

above f_{crit}, the trace alignment seems to oppose the unaliased time dip. Also apparent is that there is a different trace alignment in each of the last three panels. The lower panel shows the same filter slices applied to the event when $\Delta x = 7.5$ m, and there is no spatial aliasing.

3.8.2 *f–k* Filters in Theory, and Example

The 2D Fourier transform enables 2D filtering in much the same way as the 1D Fourier transform does in a single dimension. When applied with the *f–k* transform, such a filter is called an *f–k* filter. Usually *f–k* filters are constructed as 2D *f–k* multipliers, which can have both amplitude and phase spectra. However, just as in a single dimension, a convolution theorem can be proven to show that every *f–k* multiplier has a corresponding *t–x* impulse response that can be applied by 2D convolution with nearly the same effect. We say "nearly" rather than "exactly" because practical issues always arise, such as *t–x* aliasing caused by the use of the discrete *f–k* transform, just as time-domain aliasing arises with the DFT. In multiple dimensions, the speed advantage of the Fourier approach is usually

more telling, and convolutions are not usually done in 2D or higher dimensions unless the impulse response is very local.

Suppose $\psi(t,x)$ is a seismic gather in (t,x) space where x can refer to any spatial coordinate; then this gather is a candidate for *f–k* filtering. However, the use of the 2D DFT imposes the restriction that the x coordinate must be regularly sampled. This is usually the case for the receiver position but may not be so for other spatial coordinates such as the source position. If a coordinate is not regularly sampled, then the data must first be interpolated onto a regular grid. The basic *f–k* filter theory will be presented with continuous formulas under the assumption that the data has the regular sampling needed for the DFT.

A common data-processing need is to reject low-velocity coherent noise. Usually this corresponds to source-generated waves that are trapped in the near-surface layers, which have much lower velocities than the deeper layers. As an example, we will consider rejection of the low-velocity noise in the synthetic shot record of Figure 3.26a. This noise has an apparent velocity of $v_{\text{app}} = 1000$ m/s, while the first break velocity is about 2000 m/s. Therefore, it should be possible to design an *f–k* filter that rejects the noise event while retaining the first breaks and any events with higher apparent velocities, including the hyperbolic reflections. A common approach is to define two apparent velocities, v_{a1} and v_{a2}, and design an *f–k* multiplier that is zero between these apparent velocities and otherwise is close to unity. As the reject region is fan-shaped for both negative and positive k_x, this type of *f–k* filter is called a fan filter.

The application of an *f–k* multiplier is as simple as it sounds:

$$\hat{\psi}_m(f,k_x) = \hat{\psi}(f,kx)m(f,k_x), \tag{3.109}$$

where $m(f,k_x)$ is the filter multiplier, or mask. The filtered data in (t,x) space is recovered by an inverse *f–k* transform. The simplest possible fan filter mask is

$$m(f,k_x) = \begin{cases} 0, v_{\text{a1}} < |f/k_x| < v_{\text{a2}}, \\ 1, \text{otherwise}, \end{cases} \tag{3.110}$$

where $f, v_{\text{a1}}, v_{\text{a2}}$ are all positive and $v_{\text{a1}} < v_{\text{a2}}$. The abrupt transition from pass ($m = 1$) to reject ($m = 0$) will generally cause artifacts in the filtered result. To avoid these, a better multiplier defines a "soft" transition at the cost of making the reject region somewhat wider. Let dv be the width of the filter edge in velocity, and define $v_{\text{ao}} = v_{\text{a1}} - dv$ and $v_{\text{a3}} = v_{\text{a2}} + dv$. Then an improved fan filter multiplier is given by

$$m(f,k_x) = \begin{cases} 0, v_{\text{a1}} < |f/k_x| < v_{\text{a2}}, \\ 0.5 + 0.5\cos\left(\pi(|f/k_x| - v_{\text{ao}})/(v_{\text{a1}} - v_{\text{ao}})\right), v_{\text{ao}} < |f/k_x| \le v_{\text{a1}}, \\ 0.5 + 0.5\cos\left(\pi(|f/k_x| - v_{\text{a3}})/(v_{\text{a2}} - v_{a3})\right), v_{\text{a2}} \le |f/k_x| < v_{\text{a3}}, \\ 1, \text{otherwise}. \end{cases} \tag{3.111}$$

The second and third lines of this definition define raised-cosine tapers that transition smoothly from pass to reject or the reverse. This definition is given in terms of apparent velocity, which is constant along radial lines (lines defined by $f/k_x = \text{constant}$).

In a digital implementation, $m(f,k_x)$ must be specified on a rectangular grid in (f,k_x) space and the convergence of radial lines at the origin means that there will generally be

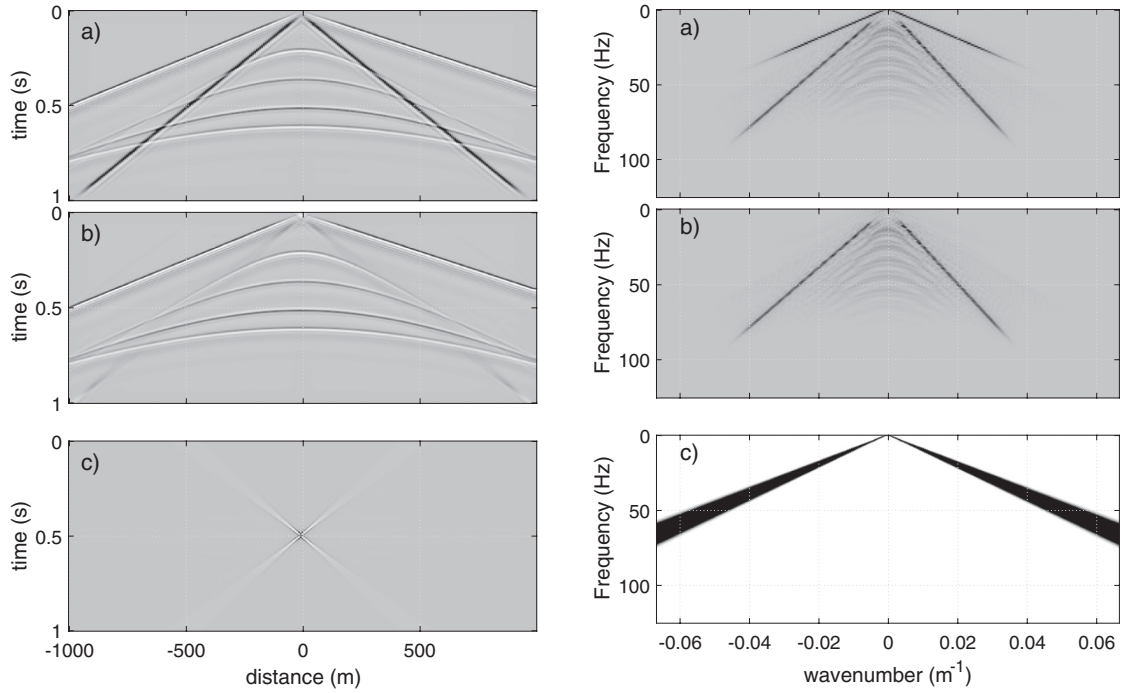


Figure 3.28a (left) The application of an *f*–*k* fan filter to the shot record of Figure 3.26a, panel a. The filter is designed to reject the 1000 m/s low-velocity noise. (a) The shot record before filtering. (b) The shot record after filtering. (c) The impulse response of the *f*–*k* fan filter.

Figure 3.28b (right) The (f, k_x) space view of the fan filtering shown in Figure 3.28a. (a) The *f*–*k* amplitude spectrum before filtering. (b) The *f*–*k* amplitude spectrum after filtering. (c) The *f*–*k* filter mask, with black indicating reject and white indicating pass.

situations where fewer than one sample lies between the velocities $v_{\mathrm{ao}}, v_{\mathrm{a1}}, v_{\mathrm{a2}}, v_{\mathrm{a3}}$ that define the filter. `fanfilter` implements an *f*–*k* fan filter and adjusts the reject region near the origin such that the first line of Eq. (3.111) is allowed to shrink to zero samples but the second and third lines are adjusted to always have at least three samples each. Thus the minimum filter width near the origin is a six-sample-wide raised-cosine taper.

Figure 3.28a shows the results of applying a fan filter to the shot record of Figure 3.26a, panel a. The filter is designed to reject the noise event, which has an apparent velocity of 1000 m/s and, given the spatial sample interval $\Delta x = 7.5$ m, is not spatially aliased. The filter was applied with `fanfilter` using $v_{\mathrm{a1}} = 950$ m/s, $v_{\mathrm{a2}} = 1050$ m/s, and $dv = 150$ m/s. Comparison of panels a and b of Figure 3.28a shows that the noise event has been almost entirely eliminated. In panel c, the impulse response of the *f*–*k* filter is shown. The same filtering effect could have been achieved by convolving this in 2D with $\psi(t, x)$. The impulse response is displayed with considerable clipping and is actually very strongly dominated by its central samples. Figure 3.28b shows the fan filtering

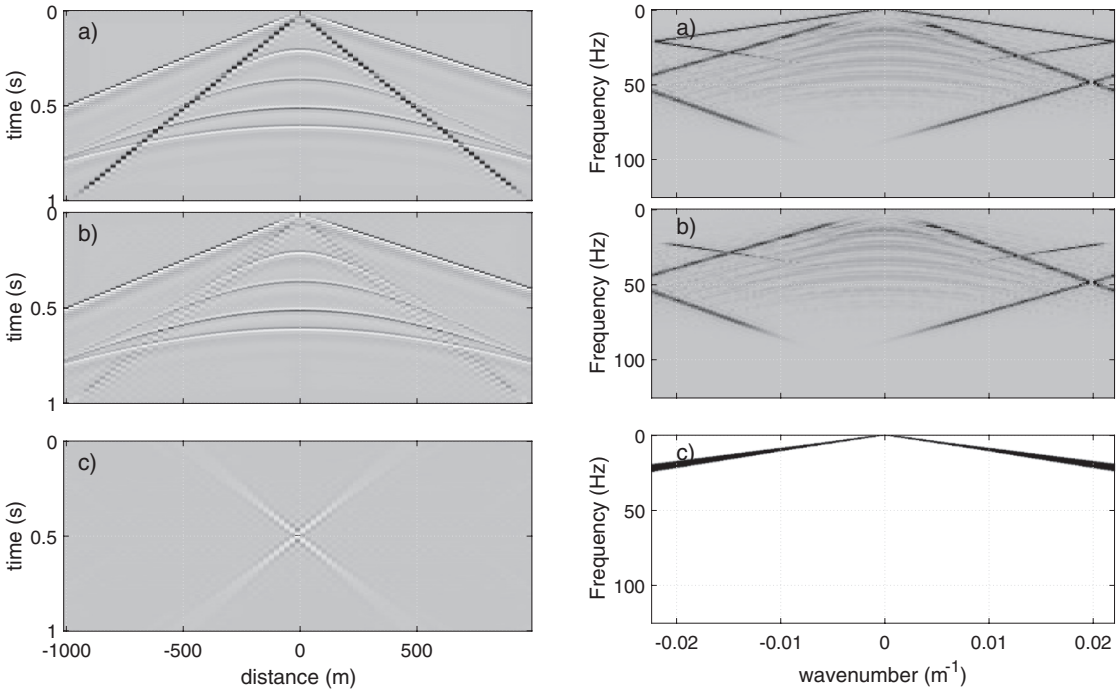


Figure 3.29a (left) The application of an f–k fan filter to the aliased shot record of Figure 3.26a, panel c. The filter is designed to reject the 1000 m/s low-velocity noise. (a) The shot record before filtering. (b) The shot record after filtering. (c) The impulse response of the f–k fan filter.

Figure 3.29b (right) The (f, k_x) space view of the fan filtering shown in Figure 3.29a. (a) The f–k amplitude spectrum before filtering. (b) The f–k amplitude spectrum after filtering. (c) The f–k filter mask, with black indicating reject and white indicating pass.

operation in the f–k domain. Comparison of panels a and b again shows that the noise event has been eliminated. Panel c shows the filter mask, as described by Eq. (3.111), where the reject region is black and the pass region is white. The two fan-shaped bands extending out from the origin to the Nyquist boundaries give the fan filter its name. The filter boundaries are very narrow near the origin but then become quite wide at the edge boundaries.

When the events to be filtered are not spatially aliased, an f–k fan filter is usually quite effective. Figures 3.29a and 3.29b show a repeat of the previous figures except that the input data was the shot record of Figure 3.26a, panel c, which has a spatial sample interval of $\Delta x = 22.5$ m and the noise event is spatially aliased. As can be seen, the filtering is less effective, although still useful, and there is a residue of the noise event that appears to be the spatially aliased part. Inspection of the f–k spectra shows that the fan filter has not been able to remove the aliased portion of the noise event. A wider reject region could be designed to do so, but this always comes at the price of causing some damage to the hyperbolic reflections, whose energy is not very localized.

3.9 Chapter Summary

The theory of sampling has been presented in two alternative ways as the link between continuous and discrete (or sampled) signals. Sampling leads necessarily to aliasing, which was described in its several manifestations. The reverse process of sampling, interpolation, was presented as the transition process from discrete to continuous signals. Next, the fundamental processes of correlation, convolution, and Fourier transformation were described in the discrete setting. The chapter closed with an introduction to time–frequency analysis with the Gabor transform and another view of the multidimensional Fourier transform, with an emphasis on understanding spatial aliasing.

4 Wave Propagation and Seismic Modeling

4.1 Introduction

Seismic wave propagation in the upper layers of the Earth's crust is a complex physical process. For example, a wave emanating from an explosive charge in a shallow (5–20 m) hole generally undergoes an immediate and rapid spectral decay before it leaves the near surface.[1] Theory predicts that this spectral decay is associated with *minimum-phase* (see Section 2.4.8) effects that produce the characteristic source waveform (Futterman, 1962). In addition to the primary downgoing pulse, there is always upgoing energy from the source that reflects off of the Earth's free surface to produce a so-called *source ghost*. In fact, there will be an entire series of similar ghosts that are produced when either the primary or a particular ghost reflects off the base of the near surface and travels up again to reflect off the free surface. Thus, it must always be expected that the downgoing pulse that serves to illuminate the subsurface will consist of a complex reverberatory series.

The subsurface is commonly idealized as a sequence of nearly homogeneous layers, called *formations*, whose geometry may be simple or complex. The contacts between the formations are called *horizons* and are usually assumed to be in *welded* contact. (Of course, slip does occur along faults in the subsurface but this is negligible over the time span of a seismic experiment.) In the case of a sedimentary basin, the formations are assumed to be horizontal and hence their material description depends only on the vertical coordinate. This book adopts the common convention of a right-handed Cartesian coordinate system with the z coordinate oriented in the vertical direction and increasing downward. Thus it is common to describe the sedimentary-basin environment as a $v(z)$ setting, by which it is meant that the propagation speed of seismic waves depends only upon the depth of formations. In any real sedimentary basin, the $v(z)$ assumption is only a first-order approximation and there are always variations in the lateral directions. These lateral variations are usually subtle in the subsurface and may be unimportant for imaging; however, in the near surface they can be very strong. The distinction between near surface and subsurface is usually drawn at the *base of weathering*, referring to the maximum depth at which weather and climate changes have an effect. Thus, a sedimentary basin may be usefully idealized as a somewhat chaotic weathered layer, 10–100 m thick and characterized by topography, low velocities, strong lateral gradients, and strong attenuation, overlying a $v(z)$ sequence of

[1] The phrase *near surface* refers to a deliberately vague region just below the Earth's surface. It is generally considered to be a region of high attenuation where static delays occur. Near-surface effects are expected to be *surface consistent*.

consolidated layers characterized by higher velocities, lower attenuation, and weak lateral gradients.

More complex settings are found near mountain belts, along passive continental margins, and near the boundaries of sedimentary basins. Here, formations can be greatly distorted from the horizontal owing to tectonic compressional or extensional forces. Common jargon for such settings is to say that they are $v(x, z)$ environments, which means that seismic wave speed can depend arbitrarily upon position.

As a wave propagates into the subsurface, a number of important physical effects occur. When a wave encounters a horizon (with different material parameters on either side), both reflected and transmitted waves result. Regardless of the type of incident wave, reflections and transmissions are generally found of each possible type of wave, called *P* or *pressure*, *S* or *shear*, and perhaps *interface* waves.[2] Even if a source can be devised to produce only one type of wave, the propagating wavefield rapidly evolves to contain all possible types going in all possible directions. For idealized elastic media and plane waves,[3] the equations governing reflection and transmission at a plane interface are known exactly. These algebraically complex equations are known as the *Zoeppritz equations* and are given by Aki and Richards (1980), and many approximate forms are also known.

Waves also lose energy as they propagate. Even in a perfectly elastic medium, a spherical wavefront suffers a $1/r$ amplitude loss that is a geometric consequence of spherical divergence (or spreading). In real media, there is always some additional loss due to anelasticity, typically quantified by the dimensionless *quality factor*, Q. These anelastic effects are often modeled by the *constant-Q theory* (Kjartansson, 1979), which refers to a Q that is independent of frequency but may vary with position. All anelastic loss theories predict that attenuation is necessarily accompanied by dispersion, or they would violate causality. Additionally, in a finely layered elastic medium, O'Doherty and Anstey (1971) showed that the cumulative effect of short-path internal multiples is a progressive attenuation of higher frequencies that is essentially indistinguishable from intrinsic Q attenuation.

These introductory remarks only hint at the complexity of the true seismic wave propagation process in the Earth's crust. They have been made here to give the reader some idea of the drastic simplifications that will be seen in the theoretical discussions in this chapter. Producing realistic numerical models of seismic wave propagation is a blend of both science and intuition. The science must be pushed to limits dictated by the available computation power, but then approximations must be made. The science can be dealt with logically; but, knowing which physical effects are small and can be neglected or developing an approximate expression for a mathematical operator is often a very subjective process.

[2] These are sometimes referred to as *primary* and *secondary* waves in Earth seismology, reflecting the fact that the P-waves arrive first and S-waves second.

[3] Plane waves are a mathematical idealization because they are necessarily infinite in extent. A point source actually emits spherical wavefronts, which may be approximately planar if the radius of the wavefront is large. A spherical wavefront may be mathematically decomposed into plane waves for analysis.

4.2 The Wave Equation Derived from Physics

There are many different "wave equations" that arise from physical theory (and many more that occur only in mathematics). The classical scalar wave equation, $\nabla^2\psi = v^{-2}\partial_t^2\psi$ (where ψ is the wave function, v is the wave propagation speed, and $\nabla^2\psi = \partial_x^2\psi + \partial_y^2\psi + \partial_z^2\psi$ is the Laplacian in 3D), is only one possibility. For example, the pressure P in an inhomogeneous fluid will be seen to obey the wave equation $\vec{\nabla}\cdot(\rho(\mathbf{x})\vec{\nabla}P) = k(\mathbf{x})\partial_t^2 P$, where $\rho(\mathbf{x})$ and $k(\mathbf{x})$ are the heterogeneous density and bulk modulus and $\mathbf{x}$ is the position vector. Though apparently quite different, what these wave equations have in common with all other wave equations is that they are *hyperbolic partial differential equations*. A great body of literature exists concerning the solution of partial differential equations, and the linear, second order equations are known to fall into three general classes: *hyperbolic*, *parabolic*, and *elliptic*. A discussion of the origin of this classification and the meaning of these terms is beyond the scope of this book and the reader is invited to consult a more general reference. There are many excellent works, and some that are our particular favorites are Morse and Feshbach (1953), Zauderer (1989), Ames (1992), and Durran (1999). An excellent, economical introductory discussion is found in Farlow (2012). These books are very practical in their orientation and cannot be said to present the subject of partial differential equations with full mathematical rigor. For this purpose, the sophisticated reader may wish to consult Gustafson (1987) or, for the very brave, Taylor (1996).

In this section, some physical systems that lead to wave equations are examined. Only systems that are relevant to the seismic exploration problem will be discussed.

4.2.1 A Vibrating String

Here we will examine the simplest second-order hyperbolic partial differential equation, which arises as the governing equation for transverse waves on a vibrating string. The presentation is adapted from that found in Morse and Feshbach (1953). The solutions to this system are an instructive beginning for the more complex problems in two and three spatial dimensions. The one-dimensional solutions can be considered as a basis for the common seismic processing step known as *stretching* that transforms a single seismic trace from time to depth or the reverse.

Consider the situation shown in Figure 4.1, where a snapshot (i.e., at constant time) of a vibrating string is shown. The displacement of the string from equilibrium, $\psi(x)$, is shown as a dashed curve and the tension, T, always acts tangentially to this curve. The tension on both ends of a differential element of the string from x to $x + dx$ creates a net force, F, that acts to restore the string to its equilibrium position. It is assumed that the magnitude of $\psi(x)$ is sufficiently small at all points that there is no significant change in T or in the length of the string. At position x in Figure 4.1, $T(x)$ is directed tangentially along the string and has a component, $T(x)\sin\theta$, that acts to restore the string to its equilibrium position. (Here, θ is the angle between the string and the horizontal.) The assumption that $T(x)$ is

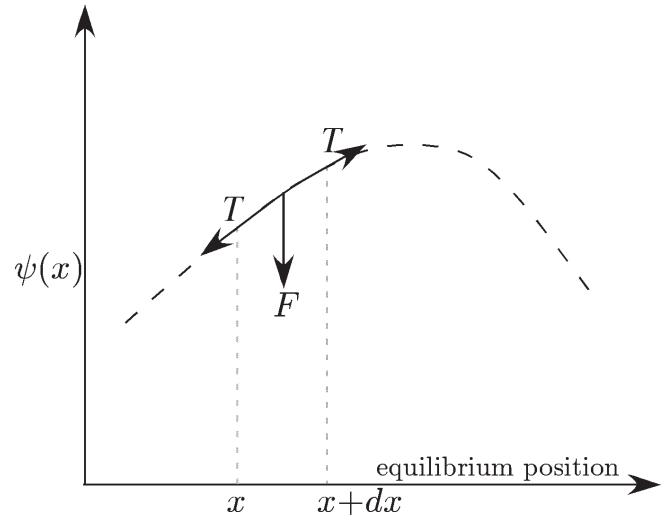


Figure 4.1 Tension, T, acts on a vibrating string whose displacement is $\psi(x)$, to produce a restoring force F.

small implies that θ is also small. This allows

$$T\sin\theta(x) \simeq T\tan\theta(x) = T\frac{\partial\psi(x)}{\partial x}, \tag{4.1}$$

where the last step follows from the geometric definition of the derivative. Considering the tension acting at both ends of the string element between x and $x + dx$ allows the total restoring force on the element to be written as

$$F(x)\,dx = T\left[\frac{\partial\psi(x+dx)}{\partial x} - \frac{\partial\psi(x)}{\partial x}\right], \tag{4.2}$$

where $F(x)$ is the force per unit length, and the minus sign in the square brackets is needed because the tension at x and $x + dx$ acts in opposite directions.

Now, recall that the definition of the second derivative of $\psi(x)$ is

$$\frac{\partial^2\psi(x)}{\partial x^2} = \lim_{dx\to 0}\frac{1}{dx}\left[\frac{\partial\psi(x+dx)}{\partial x} - \frac{\partial\psi(x)}{\partial x}\right]. \tag{4.3}$$

Thus, in the limit as dx becomes infinitesimal, Eq. (4.2) leads to

$$F(x) = T\frac{\partial^2\psi(x)}{\partial x^2}. \tag{4.4}$$

This important result says that the restoring force (per unit length) depends on the local curvature of the string. When the second derivative is positive, the curvature is concave ($\smile$), and when it is negative the curvature is convex ($\frown$). Since a positive force is directed upward in Figure 4.1 and a negative force is downward, it is easy to see that this force does act to restore the string to equilibrium.

If the string's weight is negligible, then Newton's second law (force = mass × acceleration) gives

$$T\frac{\partial^2\psi(x,t)}{\partial x^2} = \mu\frac{\partial^2\psi(x,t)}{\partial t^2}, \tag{4.5}$$

where μ is the mass per unit length and the dependence of ψ on both space and time is explicitly acknowledged. It is customary to rearrange Eq. (4.5) to give the standard form for the *one-dimensional scalar wave equation*,

$$\frac{\partial^2\psi(x,t)}{\partial x^2} - \frac{1}{v^2}\frac{\partial^2\psi(x,t)}{\partial t^2} = 0, \tag{4.6}$$

where $v = \sqrt{T/\mu}$ is the *wave velocity*.[4] This analysis has shown that the scalar wave equation arises for the vibrating string as a direct consequence of Newton's second law. Wave equations invariably arise in this context, that is, when a displacement is initiated in a continuum. Also typical is the assumption of small displacements. Generally, large displacements lead to nonlinear equations. The waves modeled here are known as *transverse* waves because the particle displacement is in a direction orthogonal to the direction of wave propagation. In Section 4.2.6, *longitudinal* waves, which have a particle oscillation in the direction of wave propagation, are discussed.

If there are external forces being applied to the string as specified by the *force density*[5] function $S(x,t)$, then Eq. (4.6) is usually modified to

$$\frac{\partial^2\psi(x,t)}{\partial x^2} - \frac{1}{v^2}\frac{\partial^2\psi(x,t)}{\partial t^2} = \frac{S(x,t)}{T}. \tag{4.7}$$

Both Eqs. (4.6) and (4.7) are examples of one-dimensional hyperbolic partial differential equations with constant coefficients. Equation (4.6) is said to be *homogeneous*, while the presence of the source term, $S(x,t)$, in Eq. (4.7) gives it the label *inhomogeneous*. Hyperbolic partial differential equations can usually be recognized right away because they involve a difference of spatial and temporal second partial derivatives being equated to a source function. Wave equations can be second order in their derivatives, as these are, or any other order as long as the highest orders of the time and space derivatives are the same. For example, a first-order wave equation known as the *advection* equation is

$$\frac{\partial\phi(x,t)}{\partial x} - a\frac{\partial\phi(x,t)}{\partial t} = 0, \tag{4.8}$$

where a is a constant.

Exercises

4.2.1 Show that the left side of Eq. (4.6) can be factored into two operators of the form of the left side of Eq. (4.8). What values does the constant a take in each

[4] It is quite common to use *velocity* for this scalar quantity even though velocity is classically a vector quantity in physics. This book conforms with this common usage.

[5] In this context, this is just a fancy way of saying force per unit length.

expression? Show that $\psi(x,t) = \psi_1(x+vt)+\psi_2(x-vt)$ is a solution to Eq. (4.6) by showing that the two factors annihilate ψ_1 or ψ_2. ($\psi_1(x+vt)$ means an arbitrary one-dimensional function of $x+vt$, and similarly for $\psi_2(x-vt)$). Can you explain the physical meaning of this result?

4.2.2 Plane Wave Solutions

In Exercise 4.2.1, a solution to the one-dimensional wave equation is given by a sum of two elementary solutions, a left-going wave and a right-going wave. For the classical homogeneous wave equation in three dimensions,

$$\nabla^2\psi = \frac{1}{v^2}\frac{\partial^2\psi}{\partial t^2}, \tag{4.9}$$

a wave can travel in many directions. Fixing a unit normal vector $\vec{n}$ and a differentiable scalar function ψ_1, it is easy to verify that the function

$$\psi(\vec{x},t) = \psi_1(\vec{x}\cdot\vec{n} - vt) \tag{4.10}$$

is a solution to Eq. (4.9). For fixed t, such a solution is constant along any plane perpendicular to the normal direction $\vec{n}$ and, as t increases, the phase of the waveform advances along the direction $\vec{n}$. For this reason, these are called plane wave solutions.

It is sometime convenient to pick a special form for the scalar function ψ_1 in Eq. (4.10), specifically a complex sinusoid, to obtain a harmonic plane wave solution,

$$\psi(\vec{x},t) = e^{i(\vec{k}\cdot\vec{x}-\omega t)}, \tag{4.11}$$

where $\vec{k}$ is the (spatial) wavenumber for a particular plane wave, measured in radians per unit distance, ω is the (temporal) frequency in radians per unit time, and the magnitudes of the two are related by the dispersion relation

$$\omega = |\vec{k}|v. \tag{4.12}$$

It will also be convenient to use the normalized form

$$\psi(\vec{x},t) = e^{2\pi i(\vec{k}\cdot\vec{x}-ft)}, \tag{4.13}$$

in which case f is the frequency (in hertz, say), and $|\vec{k}|$ is the reciprocal of the wavelength. Both forms are widely used in seismology, and familiarity with both is essential.

A general solution in terms of harmonic plane waves is obtained by integrating over all possible wavenumbers and including negative frequencies, yielding the general solution

$$\psi(\vec{x},t) = \int_{R^3} a(\vec{k})e^{i(\vec{k}\cdot\vec{x}-|\vec{k}|vt)}\,d\vec{k} + \int_{R^3} b(\vec{k})e^{i(\vec{k}\cdot\vec{x}+|\vec{k}|vt)}\,d\vec{k}, \tag{4.14}$$

where the complex-valued weights $a(\vec{k}), b(\vec{k})$ give the freedom to select a wide range of possible solutions. It is worth noting here that this integral is closely related to the inverse Fourier transform in three dimensions.

4.2.3 Spherical Solutions

It is an exercise in the chain rule to verify that the function

$$\psi(\vec{x},t) = \frac{1}{|\vec{x}|}\psi_1(|\vec{x}| - vt) + \frac{1}{|\vec{x}|}\psi_2(|\vec{x}| + vt) \tag{4.15}$$

is a solution to the homogeneous wave equation (4.9), where $|\vec{x}| = \sqrt{x^2 + y^2 + z^2}$ is the radial distance from the origin to the point $\vec{x}$. The term with ψ_1 represents a spherically symmetric wave propagating outwards from the origin, while the term with ψ_2 represents a symmetric wave traveling in toward the origin. The first term is particularly useful for modeling the propagation of a seismic wave generated by an approximate point source such as a buried dynamite charge. The $1/|\vec{x}|$ factor causes the amplitude to decay like $1/r$, where r is the radius of a sphere, while the energy on the spherical wavefront decays like $1/r^2$, inversely proportionally to the surface area of this propagating sphere.

The choice of scalar functions ψ_1, ψ_2 is quite arbitrary. So long as they are twice differentiable, they provide a spherically symmetric solution to the homogeneous wave equation with a very general choice of wave shape in the radial direction. More general solutions are obtained by translating the origin of the spherical wave to other points in space; a linear combination or weighted integral of such translates gives a general solution to the wave equation.

4.2.4 Initial Conditions for Solutions

The last two sections described very general formulations for solutions to the homogeneous wave equation, which of course include many different functions. In a real seismic wave situation, there is only one solution, so it is essential to add some additional information to select the unique solution for the situation under consideration. One common way to obtain a unique solution is by specifying the values of the function $\psi(\vec{x},t)$ and its normal derivative on some three-dimensional hyperspace in $(\vec{x},t)$ space. A formal statement of this uniqueness property, given initial conditions for the solution and its derivatives, is the content of the Cauchy–Kowalevski[6] theorem (see Taylor (1996), Vol. 1, p. 499). For seismic work, and in numerical simulations, it suffices to specify the values of $\psi(\vec{x},t_0), \partial_t\psi(\vec{x},t_0)$ at some initial time $t = t_0$. For instance, we often fix a starting time $t_0 = 0$ and specify initial conditions

$$\psi(\vec{x},0) = F(\vec{x}), \tag{4.16}$$

$$\partial_t\psi(\vec{x},0) = G(\vec{x}). \tag{4.17}$$

To see how these extra conditions select a unique solution, we apply these initial conditions to the plane wave solution in Eq. (4.14) to obtain two equations in the two unknown

[6] Also written as Cauchy–Kovalevskaya.

functions $a(\vec{k}), b(\vec{k})$ as

$$\int_{R^3} [a(\vec{k}) + b(\vec{k})] e^{i\vec{k}\cdot\vec{x}}\, d\vec{k} = F(\vec{x}), \tag{4.18}$$

$$\int_{R^3} [-a(\vec{k}) + b(\vec{k})] i|\vec{k}| v e^{i\vec{k}\cdot\vec{x}}\, d\vec{k} = G(\vec{x}). \tag{4.19}$$

After a 3D Fourier transform, this simplifies to

$$\left[a(\vec{k}) + b(\vec{k})\right] = \widehat{F}(\vec{k}), \tag{4.20}$$

$$\left[-a(\vec{k}) + b(\vec{k}))\right] = \frac{\widehat{G}(\vec{k})}{i|\vec{k}|v}, \tag{4.21}$$

which is a 2×2 system of linear equations which is easily inverted, giving a unique solution for the parameters $a(\vec{k}), b(\vec{k})$ and thus specifying a unique solution to the wave equation.

When the specified initial conditions are zero ($F \equiv 0, G \equiv 0$), the wave equation solution is of course also zero, everywhere. By including a forcing term in the wave equation, we can model a seismic source that produces a nonzero wave that grows out of the zero solutions.

For solutions to the inhomogeneous wave equation, the mathematical answer cannot be obtained as an exact integral in such a simple form. However, the Cauchy–Kowalevski theorem still applies, so specifying initial values and first derivatives at a specific time $t = t_0$ is enough to select a unique solution to the wave equation. In numerical work, we will see that it suffices to specify the initial value and a finite-difference approximation to the derivative at time $t = 0$ to obtain a unique solution.

4.2.5 Boundary Conditions

Not to be confused with initial conditions, the boundary conditions for the wave equation specify what happens to the solution when it reaches the edge of the (spatial) region of interest for the mathematical or numerical simulation. The previous three sections solved the wave equation on the infinite three-dimensional space R^3, which has no boundary. In real seismic wave modeling, waves propagate inside the finite Earth, which does have boundaries – namely the surface of the Earth. More realistically, we can only afford (computationally) to model the waves in a small region of few kilometers in diameter. It is important to specify what happens on the boundary of this region, in order to obtain a simulation that is physically reasonable.

For computational efficiency, it is common to take the region of interest to be a rectangular box, where the spatial coordinates $\vec{x}$ are restricted to the intervals $0 \le x \le L_x, 0 \le y \le L_y, 0 \le z \le L_z$. The flat surfaces of the box are the boundaries that concern us.

The simplest condition is to require the solution on a region Ω to be identically zero on the boundary $\partial\Omega$:

$$\psi(\vec{x}, t) = 0 \quad \text{for all points } \vec{x} \in \partial\Omega, \text{ and all time } t. \tag{4.22}$$

This is known as the Dirichlet condition and also the perfectly reflecting boundary condition, physically corresponding to a region whose boundaries reflect any wave back into the region without any energy loss. For a vibrating string, it corresponds to a finite-length string with each end fixed in place. At first glance, this seems like a terrible model for seismic waves as there are unlikely to be any perfect reflectors coincident with the boundary of our seismic experiment. But, since waves have a finite speed, any wave traveling from a seismic source to the edge will take some time to reach it. So, we can design the simulation with a box big enough that the waves never reach the edge in the time of the simulation.

A similar condition is to require that the solution have zero normal derivative along the boundary of the region:

$$\partial_n \psi(\vec{x}, t) = 0 \text{ for all points } \vec{x} \in \partial\Omega, \text{ and all time } t. \tag{4.23}$$

This is the Neumann condition, and corresponds to an open boundary in the acoustic wave equation. Again, this produces a large reflection at the edges, but with an inverted phase: this can be used as a suitable model for the air/Earth boundary in a seismic simulation. Note that in the case of a rectangular box, the normal derivative on an edge is just the x, y, or z partial derivative. For instance, along the boundary $x = 0$ the Neumann condition requires $\partial_x \psi(\vec{x}, t) = 0$.

Again, having all boundaries satisfy the Neumann condition is not physically realistic, but if we choose the rectangular box Ω big enough, the waves will only propagate for a short time and never hit the nonphysical boundary.

Periodic boundary conditions are obtained by assuming the solution $\psi(\vec{x}, t)$ is periodic in space, with a characteristic cell given by the box region $0 \leq x \leq L_x, 0 \leq y \leq L_y, 0 \leq z \leq L_z$. This is equivalent to forcing the solution to match values on the boundary, so that

$$\psi(0, y, z, t) = \psi(L_x, y, z, t), \tag{4.24}$$

$$\psi(x, 0, z, t) = \psi(x, L_y, z, t), \tag{4.25}$$

$$\psi(z, y, 0, t) = \psi(x, y, L_z, t). \tag{4.26}$$

Physically, this will cause a wave impinging on one edge to magically pass through to the opposite edge and travel back into the region. Not physically realistic, but, again with a big box, a simulation can run for a short time where the waves never have a chance to travel all the way to an edge. Computationally, it is sometimes advantageous to use periodic boundary conditions, for the Fourier transform then reduces to the simpler case of the Fourier series.

A final, general class to mention is that of absorbing boundary conditions. The goal here is to artificially set conditions at or near the boundary of the region of wave propagation so that when a wave reaches the boundary, very little energy is reflected back into the region of interest. This again is not physically realistic, but it only applies near the boundary and stops reflection artifacts generated by the computational boundary from contaminating the results inside the region of interest.

With a rectangular region used for the computation, one simple approach to absorption is to modify the wave equation itself across a thin layer near the edges of the rectangular region, so that the wave energy is highly attenuated as the wavefront travels toward an edge. Often this is achieved by adding a small, imaginary component to the velocity parameter in the wave equation, or by adding a dissipative term to the wave equation.

A more common approach is to impose the Clayton–Engquist boundary condition, which uses a one-way wave equation near the boundary, based on a paraxial approximation to the full acoustic or elastic equation. In two spatial dimensions x, z, attempting to absorb waves in the positive x direction, one chooses constraints in the form of one of the following:

$$\frac{\partial \psi}{\partial x} + \frac{a}{v}\frac{\partial \psi}{\partial t} = 0, \tag{4.27}$$

$$\frac{\partial \psi}{\partial z} - b\frac{\partial \psi}{\partial x} - \frac{a}{v}\frac{\partial \psi}{\partial t} = 0, \tag{4.28}$$

$$\frac{\partial^2 \psi}{\partial z \partial t} + cv\frac{\partial^2 \psi}{\partial x \partial z} - b\frac{\partial^2 \psi}{\partial x \partial t} - \frac{a}{v}\frac{\partial^2 \phi}{\partial t^2} = 0, \tag{4.29}$$

where the constants a, b, c are chosen numerically to minimize the reflections over a given range of incident angles. Similar formulas are available for three spatial dimensions, for absorption in any of the x, y, z directions.

4.2.6 Waves in a Heterogeneous Fluid

A heterogeneous fluid is a fluid whose physical properties vary with position.[7] Let $\rho(\vec{x})$ and $K(\vec{x})$ be the density and bulk modulus of such a fluid. Let $P(\vec{x})$ represent a pressure fluctuation from the ambient pressure P_0 in the fluid. Thus the total fluid pressure is $P(\vec{x}) + P_0$. The bulk modulus is a material property defined by the relation

$$P(\vec{x}, t) = -K(\vec{x})\theta(\vec{x}, t). \tag{4.30}$$

The quantity θ is the *volume strain*, or *dilatation*, and is a measure of local fractional volume change. This relation says that the pressure fluctuations are directly proportional to the induced volume changes. The volume strain θ is related to the particle velocity, $\vec{V}$, as its derivative in time is equal to the divergence of the velocity, i.e., $\partial_t \theta = \vec{\nabla} \cdot \vec{V}$. The minus sign in Eq. (4.30) is needed because an increase in pressure causes a decrease in volume, and vice versa. This is an example of a *constitutive relation* that defines how stress relates to strain. In other words, Eq. (4.30) is *Hooke's law* for a fluid.

We assume that the fluid is at rest except for those motions induced by the pressure disturbance $P(\vec{x}, t)$. As with the vibrating string, the motion of the fluid is defined by Newton's

[7] The term *inhomogeneous* is sometimes used as a synonym for "heterogeneous." However, this is in conflict with the use in Section 4.2.1, where the word "inhomogeneous" refers to a partial differential equation with a source term, and that usage has nothing to do with this.

second law,

$$\rho(\vec{x})\frac{\partial \vec{V}(\vec{x},t)}{\partial t} = -\vec{\nabla}P(\vec{x},t) + \vec{F}, \tag{4.31}$$

where $\vec{F}$ is an external force (source term) and we ignore any advective terms[8] in the time derivative of the velocity. This says that a spatially changing pressure plus any external force causes local forces in the fluid that give rise to local particle accelerations. The minus sign in front of the gradient is also necessary here, because the local forces caused by the pressure gradient point in the direction opposite to the gradient. That is, the gradient points from low to high pressure but the fluid will move from high to low pressure.

Together, the time derivative of the constitutive relation (4.30) and Newton's law (4.31) give a $3+1$-dimensional system of first-order partial differential equations, in unknowns $P, \vec{V}$, stated as

$$\frac{1}{K}\frac{\partial P}{\partial t} + \vec{\nabla}\cdot\vec{V} = 0, \tag{4.32}$$

$$\rho\frac{\partial \vec{V}}{\partial t} + \vec{\nabla}P = \vec{F}. \tag{4.33}$$

To obtain a single wave equation for the pressure, we differentiate the first equation in the system (4.32) with respect to time, and substitute in $\partial_t \vec{V}$ from the second equation in the system, to obtain

$$\frac{1}{K}\frac{\partial^2 P}{\partial t^2} - \vec{\nabla}\cdot\left(\frac{1}{\rho}\vec{\nabla}P\right) = -\frac{\vec{\nabla}\cdot\vec{F}}{\rho}. \tag{4.34}$$

The term on the right-hand side is just the external forcing term, modified for its effect on pressure.

When the density ρ is a constant, this last equation is multiplied through by ρ to obtain the classic scalar wave equation, with

$$\frac{1}{v^2}\frac{\partial^2 P}{\partial t^2} - \nabla^2 P = -\vec{\nabla}\cdot\vec{F}, \tag{4.35}$$

where $v = \sqrt{K/\rho}$ is the local velocity of sound, $\nabla^2 = \vec{\nabla}\cdot\vec{\nabla}$ is the usual Laplacian operator, and $\vec{\nabla}\cdot\vec{F}$ is the external forcing term. When ρ is not constant, applying the product rule to the term $\rho\vec{\nabla}\cdot\left((1/\rho)\vec{\nabla}P\right)$ shows that the full wave equation for the pressure is given as

$$\frac{1}{v^2}\frac{\partial^2 P}{\partial t^2} - \nabla^2 P + \vec{\nabla}\ln\rho\cdot\vec{\nabla}P = -\vec{\nabla}\cdot\vec{F}, \tag{4.36}$$

where we have used the relation $\rho\vec{\nabla}(1/\rho) = -\vec{\nabla}\ln\rho$. While this full equation is more difficult to solve, it is possible to use a solution of Eq. (4.35) to develop an approximate

[8] These are terms in the total time derivative for a fluid that are caused by fluid flowing into or out of the region of interest.

solution to the more complex Eq. (4.36). Let $\bar{P}(\vec{x}, t)$ be a solution to Eq. (4.35); then it can be used to approximately express the $\vec{\nabla} \ln \rho \cdot \vec{\nabla} P$ term in (4.36) to give

$$\frac{1}{v^2}\frac{\partial^2 P}{\partial t^2} - \nabla^2 P = -\vec{\nabla} \cdot \vec{F} - \vec{\nabla} \ln \rho \cdot \vec{\nabla} \bar{P}. \tag{4.37}$$

The last term on the right does not depend upon the unknown $P(\vec{x}, t)$ but rather plays the role of an additional source term in the scalar wave equation. Thus the effect of strong density inhomogeneity introduces an approximate source term whose strength is proportional to the logarithmic gradient of density. This process can be repeated indefinitely, with the solution from iteration $n-1$ being used to compute the effective source term for iteration n.

We can also derive a wave equation for the fluid velocity by eliminating the pressure from the system (4.32). We differentiate the second equation in the system with respect to t, and substitute $\partial P/\partial t$ from the first equation to get

$$\rho \frac{\partial^2 \vec{V}}{\partial t^2} - \vec{\nabla}(K \vec{\nabla} \cdot \vec{V}) = \frac{\partial \vec{F}}{\partial t}. \tag{4.38}$$

In the case where the bulk modulus K is constant, we divide both sides by K to obtain the classic wave equation

$$\frac{1}{v^2}\frac{\partial^2 \vec{V}}{\partial t^2} - \nabla^2 \vec{V} = \frac{1}{K}\frac{\partial \vec{F}}{\partial t}, \tag{4.39}$$

where we have used the vector calculus identity $\vec{\nabla}(\vec{\nabla} \cdot \vec{V}) = \nabla^2 \vec{V}$, which holds for irrotational fluids. As before, the parameter $v = \sqrt{K/\rho}$ is the local wave speed.

When the bulk modulus is not constant, the full wave equation becomes

$$\frac{1}{v^2}\frac{\partial^2 \vec{V}}{\partial t^2} - \nabla^2 \vec{V} = \frac{1}{K}\frac{\partial \vec{F}}{\partial t} + (\vec{\nabla} \cdot \vec{V})\, \vec{\nabla} \ln K. \tag{4.40}$$

In Cartesian coordinates, the vector Laplacian applied to the vector displacement is simply the scalar Laplacian applied separately to each displacement component. Thus this equation separates into three scalar wave equations, one for each component, with the pseudo-source term $(\vec{\nabla} \cdot \vec{V})\, \vec{\nabla} \ln K(\vec{x})$. Comparison with Eq. (4.37) shows that the density gradient determines the pseudo-source term for the pressure equation, while it is the bulk modulus gradient that does so for the displacement equation. In both equations, the wave speed is given by $v(\vec{x}) = \sqrt{K(\vec{x})/\rho(\vec{x})}$.

Exercises

4.2.2 Show that the volume dilatation θ also satisfies a scalar wave equation.

4.2.3 Show that the fluid displacement $\vec{U}$ also satisfies a vector wave equation, using the relation with volume dilatation $\theta = \vec{\nabla} \cdot \vec{U}$.

4.2.4 Consider a 1D inhomogeneous fluid. Write down the wave equation that approximately models pressure waves in a long, narrow cylinder.

4.3 Waveform Changes in Heterogeneous Wave Equation

For homogeneous media, general solutions are given in terms of sums of plane waves, which propagate individually as a single wave that translates along the direction of motion, with no change in the waveform shape or amplitude. In a heterogeneous medium, the waveform may change as it travels. As a special case, consider the one-dimensional version of the wave equation for pressure (Eq. (4.34)), which we can write as

$$\frac{1}{K}\frac{\partial^2 P}{\partial t^2} - \frac{\partial}{\partial x}\left(\frac{1}{\rho}\frac{\partial P}{\partial x}\right) = 0. \tag{4.41}$$

In the case where the impedance $K(x)\rho(x)$ is constant, no reflections are generated, so solutions are given as sums of left- and right-going waves in the form

$$P(x,t) = \psi(t \pm S(x)), \tag{4.42}$$

where the function $S(x)$ is the antiderivative of the slowness. That is,

$$S(x) = \int_{x_0}^{x} \frac{1}{v(x)}\, dx, \tag{4.43}$$

where $v(x) = \sqrt{K(x)/\rho(x)}$ is the local wave speed. The characteristic curves $t - S(x) =$ constant completely determine the solution, since the function $P(x,t)$ is necessarily constant along the characteristic curves. Thus, in a plot of these characteristic curves in the x–t plane, one can visualize the propagation of the wave as follows. It moves to the right (assuming that $\psi(t - S(x))$ is used), and the speed at which it moves is given by the reciprocal of the slope $S'(x)$ of the characteristic curves. Calling these the "slowness curves," one can deduce that a waveform compresses along a region of low velocity, and stretches out in a region of high velocity.

Referring to Figure 4.2, one sees six slowness curves (dotted lines) corresponding to times $t = 0, \ldots, 5$. An initial waveform is displayed (solid lines) along the horizontal axis $t = 0$. As time passes, the waveform moves to the right, and is displayed at times $t = 0, 5, 10, 15$. Where the slowness curves are steep, the velocity is low, and the waveform is compressed. As the slowness curves flatten out, the velocity increases and the original waveform returns.

This special substitution in 1D also works for plane waves in 3D, provided the impedance $K(x)\rho(x)$ is constant. In more general cases, more complex wave behavior occurs: reflections are generated, waveforms stretch and compress, and amplitudes are altered. Obtaining exact solutions in the general case is impossible; hence we are led to numerical methods.

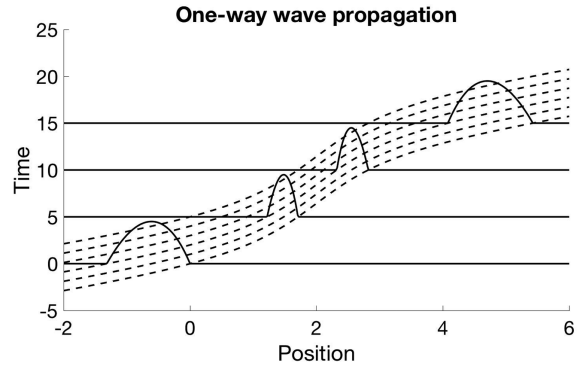


Figure 4.2 Waveform changes follow the slowness curves.

4.4 Waves in an Elastic Medium

It is a small step to go from waves in a fluid to waves in an elastic solid. Rather than just pressure as in a liquid, a solid can support local internal forces that may potentially push in any direction, against any surface in the solid. The stress tensor σ is defined at each point in the solid as a 3×3 matrix that represents the force (a 3-vector) on each of the three mutually orthogonal surfaces at that point. Newton's law tells us that the solid accelerates owing to any gradient in these internal forces, so we have a first equation of motion

$$\rho \frac{\partial^2 \vec{U}}{\partial t^2} = \vec{\nabla} \cdot \sigma + \vec{F}, \tag{4.44}$$

where ρ is the mass, $\vec{U}$ is the displacement, its second derivative is the acceleration, the del-dot of the stress is the effective internal force, and $\vec{F}$ is any externally imposed force.

The strain is the local distortion in the solid, defined as the symmetric sum of the gradients of the components of displacement, the 3×3 tensor $(\vec{\nabla}\vec{U} + \left(\vec{\nabla}\vec{U}\right)^{\mathrm{T}})/2$. The second equation needed to describe the wave motion is the constitutive relation connecting stress to strain. It can be written simply in the form

$$\sigma = \frac{1}{2} C(\vec{\nabla}\vec{U} + \left(\vec{\nabla}\vec{U}\right)^{\mathrm{T}}), \tag{4.45}$$

where C is a linear function from the tensor $\vec{\nabla}\vec{U} + \left(\vec{\nabla}\vec{U}\right)^{\mathrm{T}}$ to the tensor σ. As C is a map from a nine-dimensional space to a nine-dimensional space, it is represented by 81 coefficients, which can vary from point to point in space. A great deal of physics concerning elastic solids can be encoded into the linear function represented by C here. Particular cases of isotropic media, limited types of anisotropy such as vertical or horizontal transverse isotropy, and so on restrict the choice of C to a few numerical parameters.

In summary, the two equations (4.44) and (4.45) are all we need to model elastic wave propagation.

4.5 Finite-Difference Modeling with the Acoustic Wave Equation

This section describes a simple facility for finite-difference modeling in two spatial dimensions using the acoustic wave equation. This toolkit was originally developed by Carrie Youzwishen, who was employed by the first author for that purpose. It was subsequently evolved by this author and Dr. Zhengsheng Yao. The package provides a basic facility, using second-order finite-difference approximations, for modeling with the variable-velocity acoustic wave equation in two dimensions. Tools are provided for model building, source specification, time-stepping a wavefield, seismogram creation, and making wavefield movies. Sources and receivers can be positioned anywhere within the 2D medium (including on the boundaries), and absorbing boundary conditions are implemented. Both variable velocity and variable density can be accommodated using Eq. (4.36).

4.5.1 A Brief Introduction to Finite-Difference Approximations to Derivatives

The basic idea of finite differences is very simple. Consider the formal definition of the first derivative of a function $g(x)$, which is usually written as

$$g'(x) = \frac{dg(x)}{dx} \equiv \lim_{\Delta x \to 0} \frac{g(x + \Delta x) - g(x)}{\Delta x}. \tag{4.46}$$

If $g(x)$ is now considered to be sampled at the points $x_k = k\,\Delta x$, where k is the sample counter, then, assuming Δx to be very small, we consider the *first forward finite difference*, defined by

$$D_x^+ g(x) \equiv \frac{g(x + \Delta x) - g(x)}{\Delta x}, \tag{4.47}$$

as an approximation to the analytic first derivative. Clearly, if $g(x)$ is continuous and changes slowly relative to Δx, then this should be a good approximation to Eq. (4.46) in some sense; however, it is hardly unique, for we could equally well consider the *first backward finite difference*,

$$D_x^- g(x) \equiv \frac{g(x) - g(x - \Delta x)}{\Delta x}. \tag{4.48}$$

Comparing D_x^+ and D_x^-, we can notice that the former uses the samples $g(x)$ and $g(x+\Delta x)$ to estimate the derivative $g'(x)$, while the latter uses the samples $g(x)$ and $g(x - \Delta x)$. It is common to say that $D_x^+ g(x)$ is centered at $x + \Delta x/2$, while $D_x^- g(x)$ is centered at $x - \Delta x/2$. Thus we are trying to estimate the derivative at x with operators that are centered $\Delta x/2$ away from x. It can be shown that this introduces significant error, and it is usually preferable to develop an approximation that is centered at x. The *first centered difference*

can be constructed as the average of D_x^+ and D_x^- and is

$$D_x^0 g(x) \equiv \frac{1}{2}\left[D_x^+ g(x) + D_x^- g(x)\right] = \frac{g(x+\Delta x) - g(x-\Delta x)}{2\,\Delta x}. \tag{4.49}$$

The wave equation requires a finite-difference approximation to the second derivative. This can be done by applying two first-difference approximations in succession. Considering the four possibilities $D_x^+ D_x^+$, $D_x^- D_x^-$, $D_x^+ D_x^-$, and $D_x^- D_x^+$, only the last two will give a centered approximation, and it turns out that they are equal. So, we define the *second centered difference* as

$$D_{xx}^0 g(x) \equiv D_x^- D_x^+ g(x) = \frac{g(x+\Delta x) - 2g(x) + g(x-\Delta x)}{\Delta x^2}. \tag{4.50}$$

The last expression follows simply, as

$$\begin{aligned} D_x^- D_x^+ g(x) &= D_x^-\left[\frac{g(x+\Delta x) - g(x)}{\Delta x}\right] = \frac{1}{\Delta x}\left[D_x^- g(x+\Delta x) - D_x^- g(x)\right] \\ &= \frac{1}{\Delta x^2}\left[g(x+\Delta x) - g(x) - g(x) + g(x-\Delta x)\right] \\ &= \frac{g(x+\Delta x) - 2g(x) + g(x-\Delta x)}{\Delta x^2}. \end{aligned}$$

Equation (4.50) is commonly used for both time and space second derivatives in wave equation simulations. It is called a second-order approximation because its error is proportional to Δx^2. To see this, consider the application of D_{xx}^0 to $g(x) = e^{ikx}$, where k is a constant (the spatial frequency or wavenumber). This analysis is known as *von Neumann analysis* after its inventor. The analytic second derivative is $g''(x) = -k^2 g(x)$, so our goal is to see how closely $D_{xx}^0 g(x)$ approximates $-k^2 g(x)$. Applying Eq. (4.50) to $g(x) = e^{ikx}$ gives

$$D_{xx}^0 e^{ikx} = \frac{e^{ik(x+\Delta x)} - 2e^{ikx} + e^{ik(x-\Delta x)}}{\Delta x^2} = \frac{e^{ikx}}{\Delta x^2}\left[e^{ik\Delta x} + e^{-ik\Delta x} - 2\right]. \tag{4.51}$$

Writing $e^{ik\,\Delta x} + e^{-ik\,\Delta x} = 2\cos(k\,\Delta x)$ gives

$$D_{xx}^0 e^{ikx} = e^{ikx}\frac{2\cos(k\,\Delta x) - 2}{\Delta x^2} = e^{ikx}\left[-k^2 + k^4\Delta x^2/12 + \ldots\right], \tag{4.52}$$

where in the final expression the series expansion $\cos\theta = 1 - \theta^2/2! + \theta^4/4! - \ldots$ has been employed. So, the fractional error in $D_{xx}^0 e^{ikx}$ is

$$\left|(D_{xx}^0 g(x) - g''(x))/g''(x)\right| = k^2\,\Delta x^2/12 + \ldots,$$

confirming that the error is proportional to Δx^2, and for this reason the operator is called second order. This is important because it says that in a series of simulations run on increasingly fine grids, the error will decrease quadratically with decreasing grid size. Thus, the

error will decrease more rapidly for a fourth-order than for a second-order operator. Similar calculations show that the first centered finite difference (Eq. (4.49)) is also second order, while the forward and backward first differences are only first-order approximations. Higher-order approximations can be found in the literature and will not be derived here. A centered fourth-order second-derivative approximation is

$$D^0_{xx(4)}g(x) = \frac{-g(x+2\,\Delta x) + 16g(x+\Delta x) - 30g(x)}{12\,\Delta x^2} + \frac{16g(x-\Delta x) - g(x-2\,\Delta x)}{12\,\Delta x^2}. \tag{4.53}$$

There are two things to notice here. First, the fourth-order approximation requires five samples, compared with the three required by the second-order approximation. Second, the five samples have weights

$$[-1/12,\ 16/12,\ -30/12,\ 16/12,\ -1/12],$$

while the three points have weights of $[1,\ -2,\ 1]$. So, a higher-order approximation is less local than a lower-order one and hence requires more computer effort to apply. There are at least two ways to get greater accuracy in finite-difference simulations. One way is to implement higher-order derivative approximations, which usually means altering a computer code. The other way is to simply use whatever existing code is available and decrease the grid size.

Suppose that we wish to use the second centered difference in a simulation and we wish to have errors less than 1% with each application of the operator. A natural question to ask is, what grid size will achieve this? In order to make this question precise, we need to specify the range of wavenumbers in the simulation. We have just shown that the leading term in the relative error is $k^2\,\Delta x^2/12$, so in order to place a meaningful bound on this we need to specify the largest k of interest. Here the meaning of k is an angular one,[9] so that it relates to wavelength through $k = 2\pi/\lambda$. Now, let λ symbolize the shortest wavelength of interest, so that k is the highest wavenumber of interest. Then we write the 1% error bound as

$$\text{relative error} = \frac{k^2\,\Delta x^2}{12} = \frac{4\pi^2\,Dx^2}{12\lambda^2} < \epsilon, \tag{4.54}$$

where we will take $\epsilon = 0.01$ for the 1% error. Now, the number of samples per wavelength is $\lambda/\Delta x$, and solving for this quantity gives

$$\frac{\lambda}{\Delta x} > 2\pi\sqrt{\frac{1}{12\epsilon}}. \tag{4.55}$$

Evaluating this expression for $\epsilon = 0.01$ gives $\lambda/\Delta x >\sim 18$. So, we see that getting good performance from a finite-difference simulation requires much greater sampling than the Nyquist criterion of two samples per wavelength. A higher-order difference operator will require fewer samples per wavelength to achieve the same relative error.

[9] If we had used $g(x) = e^{i2\pi kx}$, then we would use $k = 1/\lambda$.

Exercises

4.5.1 Show that the forward and backward first difference operators have an error that is first order in the grid size. Specifically, show that the leading-order term in the absolute error, as applied to e^{ikx}, is $k^2\,\Delta x/2$.

4.5.2 Show that the first centered difference has an error term that is second order in the grid size.

4.5.3 Consider a second-derivative approximation of $D_x^0 D_x^0$. What order is this approximation? Compare it with $D_x^- D_x^+$ and discuss the pros and cons of each.

4.5.4 Show that $D_x^+ D_x^-$ and $D_x^- D_x^+$ lead to identical expressions for the second centered finite difference.

4.5.2 Wave Equation Simulations with Finite Differences

Consider the variable-velocity scalar wave equation in two dimensions,

$$\nabla^2 \psi(x,z,t) = \frac{\partial^2 \psi(x,z,t)}{\partial x^2} + \frac{\partial^2 \psi(x,z,t)}{\partial z^2} = \frac{1}{v^2(x,z)} \frac{\partial^2 \psi(x,z,t)}{\partial t^2}. \tag{4.56}$$

A second-order approximation for the time derivative may be implemented as

$$\frac{\partial^2 \psi(x,z,t)}{\partial t^2} \approx \frac{1}{\Delta t^2} \left[\psi(x,z,t+\Delta t) - 2\psi(x,z,t) + \psi(x,z,t-\Delta t)\right], \tag{4.57}$$

where Δt is the time discretization interval. Inserting this expression into Eq. (4.56) and solving for the wavefield at $t + \Delta t$ gives

$$\psi(x,z,t+\Delta t) = L_{\Delta t}\psi(x,z,t) - \psi(x,z,t-\Delta t), \tag{4.58}$$

where

$$L_{\Delta t} = \left[2 + \Delta t^2\, v^2\,(x,z)\, \nabla^2\right] \tag{4.59}$$

is the *Huygens' operator* for a time step size of Δt.

Equation (4.58) is an expression for *time-stepping* the wavefield. It shows that estimation of the wavefield at $t + \Delta t$ requires knowledge of the two earlier wavefields at t and $t - \Delta t$. Each of these wavefields is called a *snapshot* and, in a 2D computer simulation, they are two-dimensional matrices. In this expression, only the time derivative has been made finite, while the spatial derivatives are left abstract as symbolized by ∇^2. This allows a uncomplicated view of the essence of a time-stepping method. Given the snapshots $\psi(x,z,t)$ and $\psi(x,z,t-\Delta t)$, a single time step is accomplished by first operating on $\psi(x,z,t)$ with the operator $L_{\Delta t} = \left[2 + \Delta t^2\, v^2(x,z)\, \nabla^2\right]$ and then subtracting $\psi(x,z,t-\Delta t)$ from this. This process is illustrated in Figure 4.3a. Essentially, the time step can be understood as a direct application of Huygens' principle. Attributed to Christiaan Huygens in his 1678 *Traité de Lumiere*, the construction of a wavefield Δt in the future is done by replacing each point in the wavefield at time t_0 by a spherical "Huygens wavelet" of radius

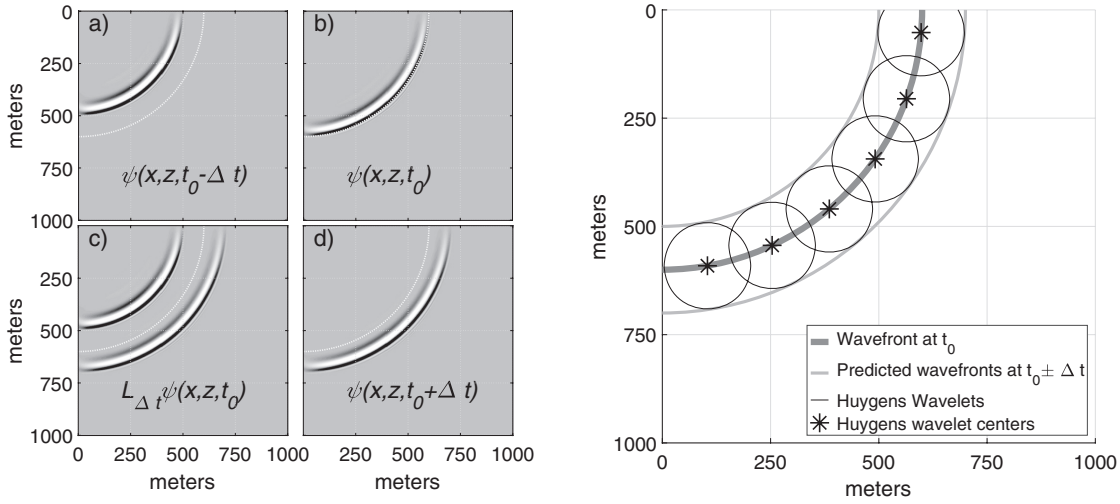


Figure 4.3a (left) Computation of a single finite-difference time step. A source at $(x, z) = (0, 0)$ emits a minimum-phase wavelet in a constant-velocity medium. In each panel, the dotted white line shows the theoretical position of the wavefront at $t = t_0$. (a) The snapshot of the wave at $t = t_0 - \Delta t$. (b) The snapshot of the wave at $t = t_0$. (c) The result of the calculation of $L_0\psi(x, z, t_0)$. Note that the wave is propagated both forward and backwards, corresponding to the traveltime Δt. (d) The snapshot of the wave at $t = t_0 + \Delta t$, calculated by subtracting the snapshot in panel a from the field shown in panel c. See Eq. (4.58) and Figure 4.3b.

Figure 4.3b (right) A Huygens' principle construction is illustrated. Each point on the wavefront at t_0 is replaced by a circular "Huygens wavelet" of radius $r_h = v\,\Delta t$. The superposition of all such Huygens wavelets predicts the wavefronts at $t_0 + \Delta t$ and $t_0 - \Delta t$. This is the action of the operator $L_{\Delta t}$ given by Eq. (4.59). Compare with Figure 4.3a.

$r = v\,\Delta t$, and the superposition of infinitely many such wavelets gives the wavefield in the future. Huygens specifically stated that the wavelets should only be constructed in the direction of forward wave propagation; however, the actual construction (Figure 4.3b) uses a full 360°wavelet and predicts both the forward- and backward-propagated wavefronts. The operator $L_{\Delta t}$ given by Eq. (4.59) achieves this construction, as shown in Figure 4.3a, part c. The wavefield Δt in the future is then isolated by subtraction of the wavefield Δt in the past as detailed in Eq. (4.58). Figure 4.3a is actually a considerable exaggeration designed to make the construction obvious. It really only works for a very small Δt, so small that the highest frequency in the wave has a period 10 or 20 times greater than Δt. If Figure 4.3a were made for such a small time step, it would be exceedingly difficult to visually distinguish the four snapshots in the figure.

Equation (4.58) shows that $\psi(x, z, t + \Delta t)$ is estimated from the superposition of three terms: $2\psi(x, z, t)$, $\Delta t^2\, v^2(x, z)\, \nabla^2\psi(x, z, t)$, and $-\psi(x, z, t - \Delta t)$. Of these, the second requires the computation of the Laplacian (∇^2) of the current wavefield, which is an expensive operation. MATLAB supplies the function `del2`, which computes $0.25\nabla^2$ of a matrix using centered second-order finite operators that are modified near the boundary. Experimentation showed that `del2` was not optimal for use with absorbing boundary conditions, so two alternate Laplacians, *`del2_5pt`* and *`del2_9pt`*, were created. Both of

these functions pad the entire boundary of the input matrix with extra rows and columns of zeros, and this works better with absorbing boundaries. `del2_5pt` implements ∇^2 using second-order finite-difference operators, while `del2_9pt` uses fourth-order operators. Thus `del_5pt` computes the approximation

$$\nabla^2\psi(x,z,t) = \frac{\psi(x+\Delta x,z,t) - 2\psi(x,z,t) + \psi(x-\Delta x,z,t)}{\Delta x^2} + \frac{\psi(x,z+\Delta z,t) - 2\psi(x,z,t) + \psi(x,z-\Delta z,t)}{\Delta z^2}, \tag{4.60}$$

and `del2_9pt` computes

$$\begin{aligned}\nabla^2\psi(x,z,t) &= \frac{1}{12\,\Delta x^2}\left[-\psi(x+2\,\Delta x,z,t) + 16\psi(x+\Delta x,z,t) - 30\psi(x,z,t)\right] \\ &\quad + \frac{1}{12\,\Delta x^2}\left[16\psi(x-\Delta x,z,t) - \psi(x-2\,\Delta x,z,t)\right] \\ &\quad + \frac{1}{12\,\Delta z^2}\left[-\psi(x+2\,\Delta z,z,t) + 16\psi(x+\Delta z,z,t) - 30\psi(x,z,t)\right] \\ &\quad + \frac{1}{12\,\Delta z^2}\left[16\psi(x-\Delta z,z,t) - \psi(x-2\,\Delta z,z,t)\right].\end{aligned} \tag{4.61}$$

The core function in the finite-difference toolbox is `afd_snap` (the "`afd`" prefix stands for "acoustic finite difference"). The function `afd_snap` requires two input wavefields, representing $\psi(x,z,t)$ and $\psi(x,z,t-\Delta t)$, and computes $\psi(x,z,t+\Delta t)$ according to Eq. (4.58). This function is intended to be used in a computation loop that time-increments a wavefield by any number of steps. Two initial wavefields must be constructed to start the simulation, and sources may be prescribed by placing appropriate impulses in these two wavefields. Receivers are simulated by extracting samples at each receiver location from each $\psi(x,z,t)$ as it is computed. These extracted samples may be accumulated in vectors representing the recorded traces.

The use of Eq. (4.58) in a time-stepping simulation is known to be unstable in certain circumstances (e.g., Mitchell and Griffiths (1980)). Instability means that the amplitudes of the wavefield grow without bound as it is stepped through time. The key to this behavior is the amplitude of the $\nabla^2\psi(x,z,t)$ term in Eq. (4.58). Using the five-point Laplacian of Eq. (4.60) (with $\Delta z = \Delta x$) in Eq. (4.58) leads to

$$\psi(x,z,t+\Delta t) = 2\psi(x,z,t) - \psi(x,z,t-\Delta t) + \frac{\Delta t^2\, v^2}{\Delta x^2}\left[\delta_{xx}\psi(x,z,t) + \delta_{zz}\psi(x,z,t)\right], \tag{4.62}$$

where $\delta_{xx}\psi(x,z,t) = \psi(x+\Delta x,z,t) - 2\psi(x,z,t) + \psi(x-\Delta x,z,t)$ and similarly for δ_{zz}. In this expression, all of the ψ terms can be considered to have similar magnitude. Thus, the factor $\Delta t^2\, v^2\, \Delta x^{-2}$ is a possible *amplification factor* if it becomes too large. Lines et al. (1999) show that the condition for stability is

$$r \equiv \frac{v\,\Delta t}{\Delta x} \le \frac{2}{\sqrt{a}}, \tag{4.63}$$

where the constant a is the sum of the absolute values of the weights for the various wavefield terms in the finite-difference approximation for ∇^2. The constant $r \equiv v\,\Delta t/\Delta x$ is called the *Courant number* and plays an important role in wave-equation simulations. In a variable-velocity simulation, the fastest velocity found anywhere in the model is used to calculate r so that stability can be assured throughout. For the Laplacian of Eq. (4.60), $a = 8$, while for Eq. (4.61), $a = 128/12 = 32/3$. Also, since v is a function of (x, z), it suffices to use the maximum velocity in the model. Thus the stability conditions are

$$r = \frac{v_{\max}\,\Delta t}{\Delta x} \leq \begin{cases} \dfrac{1}{\sqrt{2}} & \text{second-order Laplacian,} \\ \sqrt{\dfrac{3}{8}} & \text{fourth-order Laplacian.} \end{cases} \tag{4.64}$$

Thus, the time and space sample rates cannot be chosen independently. Generally, finite-difference operators need many more samples than the Nyquist criterion of two per wavelength. Technically, this is because the operators cause an artificial dispersion called *grid dispersion*. Grid dispersion preferentially affects the shorter wavelengths, so oversampling reduces the dispersion. A good rule of thumb is about 5–10 samples per wavelength. Typically, in the creation of a model, a desired temporal frequency range is known. Then, the minimum wavelength is given by $\lambda_{\min} = v_{\min}/f_{\max}$ and the spatial sample rate can be chosen to achieve a desired number of samples per wavelength. Finally, the temporal sample rate is chosen to achieve stability.

Usually, the user will not invoke `afd_snap` directly. Instead, `afd_shotrec` is provided to create a source record and `afd_explode` will create exploding reflector models. The exploding reflector model and `afd_explode` will be described in Chapter 7, and only `afd_shotrec` will be discussed here. `afd_shotrec` requires inputs giving the temporal and spatial sample sizes, the maximum record time, the velocity matrix, the receiver positions, the wavelet, the desired Laplacian (five-point or nine-point), and the two initial snapshots of the wavefield (`snap1` and `snap2`). The snapshots should be initialized to matrices of zeros of the same size as the velocity model. Then the source configuration is described by placing appropriate impulses in these two snapshots. A simple strategy is to leave `snap1` as all zeros and simply place impulses at the source locations in `snap2`.

Code Snippet 4.5.1 illustrates the use of these finite-difference facilities to model a single shot record. First, a three-layer velocity model is defined in lines 1–11 by calling `afd_vmodel` twice. The velocity model is a matrix representing P-wave velocity in (x, z) space. The function `afd_vmodel` is a convenience function that fills in a polygonal area in a matrix with a constant value. The polygonal area is defined by the (x, z) coordinates of its nodes, and the coordinate origin is assumed to be the upper left corner of the model. In all of these finite-difference codes, the vertical and horizontal sample sizes must be identical. In this example, the velocity model has three layers with horizontal interfaces and layer velocities of 2000, 2800, and 3200 m/s.

The model is built by first defining (x, z) coordinates (lines 2 and 3) and filling the velocity matrix with 3200 m/s (line 5). Subsequently, two polygons are defined to represent the two upper layers. On line 6, `z1` and `z2` are the depths to the bottom of the first and second layers. Line 8 defines the (x, z) coordinates of the four vertices of a rectangle

Code Snippet 4.5.1 This code illustrates the use of *afd_shotrec* to model a shot record. Lines 1–12 build the velocity model, and lines 14–26 create a second-order and a fourth-order seismogram. The results are shown in Figures 4.4, 4.5a, and 4.5b.

```
1   %make a velocity model
2   nx=128;dx=10;nz=128; %basic geometry
3   x=(0:nx-1)*dx;z=(0:nz-1)*dx;
4   v1=2000;v2=2800;v3=3200;%velocities
5   vmodel=v3*ones(nx,nz);
6   z1=(nz/8)*dx;z2=(nz/2)*dx;
7   dx2=dx/2;
8   xpoly=[-dx2 max(x)+dx2 max(x)+dx2 -dx2];
9   zpoly=[-dx2 -dx2 z1+dx2 z1+dx2];
10  vmodel=afd_vmodel(dx,vmodel,v1,xpoly,zpoly);
11  zpoly=[z1+dx2 z1+dx2 z2+dx2 z2+dx2];
12  vmodel=afd_vmodel(dx,vmodel,v2,xpoly,zpoly);
13
14  dtstep=.001;%time step
15  dt=.004;tmax=1;%time sample rate and max time
16  xrec=x;%receiver locations
17  zrec=zeros(size(xrec));%receivers at zero depth
18  snap1=zeros(size(vmodel));
19  snap2=snap1;
20  snap2(1,length(x)/2)=1;%place the source
21  %second order laplacian
22  [seismogram2,seis2,t]=afd_shotrec(dx,dtstep,dt,tmax, ...
23          vmodel,snap1,snap2,xrec,zrec,[5 10 30 40],0,1);
24  %fourth order laplacian
25  [seismogram4,seis4,t]=afd_shotrec(dx,dtstep,dt,tmax, ...
26          vmodel,snap1,snap2,xrec,zrec,[5 10 30 40],0,2);
```

End Code

wavepropcode/afd_example1.m

representing the first layer. A property of the algorithm used by *afd_vmodel* is that points that lie exactly on the boundary of the polygon are considered *outside* the polygon and so do not acquire the new velocity. The variable `dx2`, defined on line 7 as half of the grid spacing, is used in line 8 to define a rectangle that is half a grid spacing above depths 0 and `z2` and half a grid spacing outside the minimum and maximum x coordinates. This ensures that the rectangle extends to the limits of the velocity matrix. Line 9 fills this rectangle with the velocity of 2000 m/s and then lines 10 and 11 repeat this process for the next layer. The resulting velocity model is shown in Figure 4.4. This plot was made using `plotimage(vmodel-3000,z,x)`. A constant is subtracted from the velocity model so that the resulting matrix has both positive and negative values, as expected by *plotimage*. The raypaths shown in this figure correspond to traveltimes shown in Figures 4.5a and 4.5b.

Code Snippet 4.5.1 creates two seismograms: the first (line 21) uses a second-order Laplacian (Eq. 4.60) and the second (line 23) uses a fourth-order Laplacian (Eq. 4.61). The preparation for the seismograms defines the time step (line 13), the temporal sample

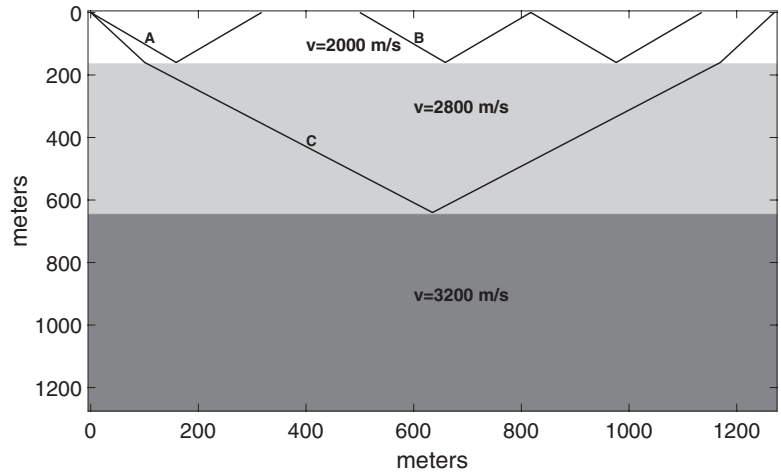


Figure 4.4 The velocity model created in Code Snippet 4.5.1. Three raypaths are also shown: A: a primary reflection off the first interface; B: a first-order multiple in the top layer; C: a primary reflection off the second interface.

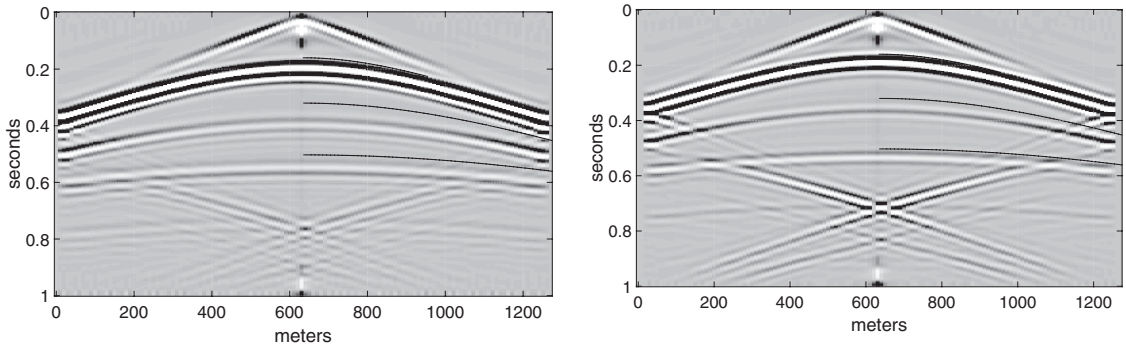


Figure 4.5a (left) The second-order seismogram created on line 21 of Code Snippet 4.5.1. Superimposed on the right-hand side are ray-traced traveltimes for (from top to bottom) the first primary reflection, the first-order multiple in the top layer, and the second primary reflection. The corresponding raypaths are shown in Figure 4.4.

Figure 4.5b (right) The fourth-order seismogram created on line 23 of Code Snippet 4.5.1. See Figure 4.5a for a description of the superimposed traveltimes.

rate (line 14), the maximum time (line 14), the receiver locations (lines 15 and 16), and the source strength and geometry (line 19). The time step is generally chosen to be small enough to satisfy the stability requirement (Eq. 4.64), and the temporal sample rate is usually much more coarse. `afd_shotrec` internally calculates the seismogram at the sample rate of the time step and then resamples it to the desired temporal sample rate. This is sensible because it is a well-known property of finite-difference methods that the higher frequencies are physically inaccurate. In this case, the Nyquist frequency for a sample rate of 0.001 s is 500 Hz, while for 0.004 s it is 125 Hz. Considering that the antialias filter for

resampling to 0.004 s will attenuate frequencies above about half of Nyquist, this example anticipates using only frequencies below about 60 Hz, and that is only about 10% of the original Nyquist frequency.

The variables `snap1` and `snap2` created on lines 17–19 represent the wavefields $\psi(x, z, t = -\Delta t)$ and $\psi(x, z, t = 0)$, respectively. They are created to be the same size as the velocity model, and `snap1` is a matrix of zeros. The source is specified by placing appropriate nonzero samples in `snap2`. In this case, a point source in the center of the x axis at $z = 0$ is simulated.

At this stage, all of the input parameters to `afd_shotrec` have been described except for the final three. These specify the filter (or wavelet) to be convolved with the seismogram, the phase of the filter, and the order of the Laplacian. In this case, *Ormsby* filter specifications are given as [5 10 30 40] and this means that the filter passband begins at 5 Hz, reaches full amplitude at 10 Hz, begins to ramp down at 30 Hz, and rejects frequencies above 40 Hz. The penultimate parameter specifies the phase of the filter, which is, in this case, zero (a value of 1 gives *minimum phase*). The last parameter is either 1, indicating a second-order Laplacian, or 2, meaning a fourth-order approximation.

The resulting seismograms are shown in Figures 4.5a and 4.5b, respectively. Both of these figures are plotted with a highly clipped display, otherwise the direct wave near the source would be the only visible arrival. Though similar at first glance, these two seismograms are significantly different. The three hyperbolic reflections in each figure are (from the top) the primary reflection off the bottom of the first layer, the first-order multiple reflecting between the bottom of the first layer and the surface, and the primary reflection off the bottom of the second layer. The traveltimes for these events are superimposed on the right-hand sides of the figures, and their raypaths are shown in Figure 4.4. The reflections for these events lag significantly behind the ray-traced traveltimes, but more so in the second-order solution. This time lag occurs because the finite-difference approximations to the derivative in the wave equation have a frequency-dependent performance. Essentially, for wavelengths that are large compared with the computation grid, the approximate derivatives are acceptable but, as the wavelength approaches the grid spacing, the approximation becomes very poor. The result is a phenomenon known as grid dispersion, meaning that the actual propagation speed of waves on the grid is not simply $v_{\text{ins}}(x, z)$ but is a complex function of wavelength (or, equivalently, frequency). Grid dispersion makes the reflections appear to lag behind the corresponding raytrace traveltimes. It also makes the apparent wavelet appear more elongated (dispersed). Comparison of the first reflection at far offsets on both seismograms shows that the second-order result has a more dispersed waveform. Also, the fourth-order result has a smaller lag behind the ray-trace traveltimes than the second-order result for all reflections.

Careful inspection of Figures 4.5a and 4.5b shows a set of events that originate where each reflection meets the sides of the seismogram and then cross in the center of the figures. These are typical artifacts known as *edge effects*. They arise because a computer simulation of wave propagation must always operate in a finite domain with definite boundaries. When a wave encounters such a boundary, it behaves as though it has met a perfect reflector. These boundary reflections would be much worse if `afd_shotrec` did not incorporate *absorbing boundaries* (Clayton and Engquist, 1977). Obviously, absorbing

boundaries are not perfect, but they do help. The boundary-related artifacts can be seen to be generally quite steeply dipping in (x, t). This is because the absorbing boundary conditions are optimized for a small range of wavefront incidence angles around normal incidence. That is, a wavefront that encounters the boundary at normal incidence is completely absorbed, while one making an angle of, say, 30° is partially reflected.

4.6 The One-Dimensional Synthetic Seismogram

For a one-dimensional acoustic medium, there exists an algorithm that can generate a complete synthetic seismogram that includes all multiples and the effects of transmission losses. The theory is also very flexible in that it allows the explicit separation of the various physical effects. These 1D synthetic seismograms are a very important tool in seismic interpretation. After migration, or for nearly horizontal reflectors in any case, the 1D model is quite appropriate. Comparison of 1D synthetic seismograms with seismic data, a process called *tying*, allows reflections from key geologic markers to be identified. Typically, velocity and density information is provided through *well logging* as finely sampled functions of depth. A common sample rate is 1 foot, or 0.31 m. The well logging is usually confined to a certain depth zone beginning at the exploration target and extending upwards perhaps hundreds of meters. The near surface is rarely logged and is usually represented as a homogeneous layer whose velocity is selected to optimize the tie between the synthetic seismogram and the seismic data. The theory presented here constructs the response of the 1D medium to a unit impulse, the *impulse response*, and this must be convolved with an appropriate wavelet before comparison with seismic data. The estimation of the appropriate wavelet is a potentially complex task that is called *wavelet analysis*.

4.6.1 Normal-Incidence Reflection Coefficients

Consider a 1D displacement wave incident from above upon an interface where v_1, ρ_1 and v_2, ρ_2 are the velocities and densities of the upper and lower media, respectively. Let the incident wave be described by $f(t - z/v_1)$, the reflected wave by $g(t + z/v_1)$, and the transmitted wave by $h(t - z/v_2)$. Here f, g, and h represent arbitrary functions describing traveling waves. The reflected and transmitted waves can be determined in terms of the incident waves by requiring that the total wavefield satisfy *continuity of displacement* and *continuity of pressure*. The first condition is required so that the interface remains in welded contact during the passage of the wave. The second condition is needed to ensure that the local acceleration remains finite at the interface. This is because the local force is determined by the pressure gradient and a discontinuous pressure means an infinite force, and that means infinite acceleration.

For definiteness, let the interface be at $z = 0$, and then continuity of displacement is expressed by

$$f|_{z=0} + g|_{z=0} = h|_{z=0}. \tag{4.65}$$

To develop a form for continuity of pressure, Hooke's law can be used. As shown in Eq. (4.30), for an inhomogeneous fluid, Hooke's law relates the applied pressure to the negative of the differential volume change. In one dimension, this reduces to $p = -K\,\partial_z u$, where p is the pressure, u is the displacement, and K is the bulk modulus. In the development of Eq. (4.35), it was shown that the wave velocity is given by the square root of the bulk modulus divided by the density or, equivalently, $K = \rho v^2$. Thus continuity of pressure can be expressed by

$$\rho_1 v_1^2 \left.\frac{\partial f}{\partial z}\right|_{z=0} + \rho_1 v_1^2 \left.\frac{\partial g}{\partial z}\right|_{z=0} = \rho_2 v_2^2 \left.\frac{\partial h}{\partial z}\right|_{z=0}. \tag{4.66}$$

Since $f = f(t - z/v_1)$, then $\partial_z f = -v_1^{-1} f'$, where f' is the derivative of f with respect to its entire argument. With similar considerations for g and h, Eq. (4.66) becomes

$$\rho_1 v_1 f'|_{z=0} + \rho_1 v_1 g'|_{z=0} = \rho_2 v_2 h'|_{z=0}, \tag{4.67}$$

which can be immediately integrated to give

$$\rho_1 v_1 f|_{z=0} + \rho_1 v_1 g|_{z=0} = \rho_2 v_2 h|_{z=0}. \tag{4.68}$$

Considering the incident wavefield as known, Eqs. (4.65) and (4.68) constitute two linear equations for the two unknowns g and h. Defining the *acoustic impedance* $I = \rho v$, the solution can be written as

$$\begin{aligned} g|_{z=0} &= -Rf|_{z=0}, \\ h|_{z=0} &= Tf|_{z=0}, \end{aligned} \tag{4.69}$$

where the *reflection coefficient*, R, and the *transmission coefficient*, T, are given by

$$R = \frac{I_2 - I_1}{I_2 + I_1}, \tag{4.70}$$

$$T = \frac{2I_1}{I_2 + I_1}. \tag{4.71}$$

The reflection coefficient is defined such that a transition from lower to higher impedance gives a positive R. It is easily shown that $R + T = 1$. A wave incident from below on the same interface will experience a reflection coefficient of $-R$ and a transmission coefficient of $I_2 T/I_1 = 1 + R$. As the first of the equations (4.69) shows, the reflected wave is actually $-R$ times the incident wave. This is because these are displacement waves and the direction of particle displacement reverses at a positive impedance contrast. This can be understood by considering the limiting case of $I_2 \to \infty$, corresponding to incidence upon a perfectly rigid material. In this case, $T \to 0$ and continuity of displacement requires that the reflected wave have an equal and opposite displacement to that of the incident wave.

Exercises

4.6.1 By directly computing pressures, show that the reflected pressure is $+R$ times the incident pressure. Also, show that the transmitted pressure is $+T$ times the incident pressure. Thus the displacement and pressure waves have the same transmission equation but there is a sign difference in their reflection expressions. Explain these results with a physical description. (Hint: If the particle displacement is in the same direction as the wave propagation, then is the pressure a compression or a rarefaction? What if the displacement is in the opposite direction to the direction of wave propagation? Is a compression a positive or negative pressure? Why?)

An alternative form for the reflection and transmission coefficients involves writing them in terms of the contrast and average of the impedance across the layer. Define $I = (I_1 + I_2)/2$ and $\Delta I = I_2 - I_1$, and Eq. (4.69) becomes

$$R = \frac{\Delta I}{2I},$$
$$T = \frac{I - 0.5\,\Delta I}{I}. \tag{4.72}$$

Using $I = \rho\alpha$ allows R to be reexpressed as

$$R = \frac{1}{2}\left[\frac{\Delta v}{v} + \frac{\Delta \rho}{\rho}\right], \tag{4.73}$$

which expresses the contributions of both v and ρ to the reflection coefficient.

4.6.2 A "Primaries-Only" Impulse Response

Consider a layered 1D medium described by a wave velocity $v(z)$ and a density $\rho(z)$ for $z \geq 0$. In order to apply the results of the previous section, let $\alpha(z)$ and $\rho(z)$ be approximated by discrete sequences $[v_k]$ and $[\rho_k]$, $k = 1, 2, \ldots, N$, representing a stack of homogeneous layers having thicknesses of $[\Delta z_k]$. Furthermore, let the thicknesses be chosen such that

$$\frac{\Delta z_k}{v_k} = \frac{\Delta t}{2} = \text{constant}, \tag{4.74}$$

where Δt is a constant giving the two-way traveltime across any layer. (Models with constant-traveltime layers are usually called *Goupillaud models*, after Goupillaud (1961).) This requirement is not strictly necessary for the theory but is convenient for algorithm development. The condition means that homogeneous intervals of high velocity will be represented by more samples than corresponding intervals of low velocity. It is quite acceptable to have many layers with nearly zero impedance contrast. By choosing Δt sufficiently small, $v(z)$ and $\rho(z)$ can usually be approximated with sufficient accuracy.

The constant Δt becomes the temporal sample rate of the output seismogram. However, greater realism is obtained by choosing Δt much smaller than the desired sample rate and resampling the seismogram after it is computed.

Given this discrete approximation to the desired model, it is possible to explicitly calculate all primary reflections. For each layer, let the sequence $[R_k]$ denote the reflection coefficient at the layer bottom for incidence from above. Let a coincident source–receiver pair be placed at $z = 0$ and let the source emit a unit impulse of *displacement* at $t = 0$. The reflection from the first interface will have magnitude $-R_1$ and will arrive at the receiver at time Δt. The minus sign arises from Eq. (4.69) and occurs because the reflection reverses the direction of displacement. A z-transform expression conveniently expresses the first arrival as $-R_1 z^{\Delta t}$. The transmitted wave, of amplitude $1 - R_1$, crosses the first interface and reflects with reflection coefficient $-R_2$. It recrosses the first interface from below with a transmission coefficient of $1 + R_1$ and then arrives at the receiver. This second reflection has a z-transform of $-(1-R_1)R_2(1+R_2)z^{2\,\Delta t} = -(1-R_1^2)R_2 z^{2\,\Delta t}$, so the net transmission effect of two-way passage through the first interface is $1 - R_1^2$. Similar reasoning leads to the z-transform of the third reflection as $-(1 - R_1^2)(1 - R_2^2)R_3 z^{3\,\Delta t}$, and for the general jth arrival $-\prod_{k=1}^{j-1}(1-R_k^2)R_j z^{j\,\Delta t}$. A primaries-only synthetic seismogram of displacement for N reflectors is represented by the sum of these z-transform terms as

$$u_{\text{imp}}(z) = 1 - R_1 z^{\Delta t} - (1 - R_1^2)R_2 z^{2\,\Delta t} - (1 - R_1^2)(1 - R_2^2)R_3 z^{3\,\Delta t} - \ldots - \left[\prod_{k=1}^{j-1}(1 - R_k^2)\right] R_j z^{j\,\Delta t} - \cdots - \left[\prod_{k=1}^{N-1}(1 - R_k^2)\right] R_n z^{n\,\Delta t}, \quad (4.75)$$

where the subscript "imp" indicates "impulse response" and the leading 1 represents a direct arrival. This expression has the compact representation

$$u_{\text{imp}}(z) = 1 - \sum_{j=1}^{N}\left[\prod_{k=1}^{j-1}(1 - R_k^2)\right] R_j z^{j\,\Delta t}. \quad (4.76)$$

The term in square brackets is called the *transmission loss*. Often seismic data has had a correction applied for transmission loss, in which case it is useful to construct a synthetic seismogram without transmission losses by setting these terms to unity. The result is the simplest model of a seismogram as reflection coefficients arriving at the two-way traveltime to each reflector.

A source waveform other than a unit impulse is easily accounted for. If $W(z)$ represents the z-transform of the source waveform, then, for example, the second reflection is given by $(1 - R_1^2)R_2 W(z) z^{2\,\Delta t}$. That is, $W(z)$ simply multiplies the z-transform of the impulsive arrival. For example consider a causal waveform with $W(z) = w_0 + w_1 z^{\Delta t} + w_2 z^{2\,\Delta t} + \ldots$. Then the reflection and transmission effects scale each sample of $W(z)$ equally and the later samples simply arrive superimposed on top of later reflections. Of course, this is just a physical way of visualizing a convolution. Thus it is concluded that the source waveform is included in the synthetic seismogram by convolving it with the impulse response (e.g.,

Eq. (4.76)) of the layered medium. Written with z-transforms, this is simply

$$S(z) = u_{\text{imp}}(z)W(z). \tag{4.77}$$

It may also be of interest to compute a *pressure* seismogram, as this models what is measured by a hydrophone. As discussed in Exercise 4.6.1, the essential difference is that the pressure does not change sign upon reflection as pressure does. The transmission effect is the same for both pressure and displacement. Thus, for the pressure response to a unit impulse of pressure, Eq. (4.69) should be modified to read

$$p_{\text{imp}}(z) = 1 + \sum_{j=1}^{N} \left[\prod_{k=1}^{j-1} (1 - R_k^2) \right] R_j z^{j\,\Delta t}. \tag{4.78}$$

4.6.3 Inclusion of Multiples

Waters (1981) gives a computational scheme that allows the calculation of all primaries and multiples up to any desired order. The method appears to be based on the earlier proposal of Goupillaud (1961), who suggested that the Earth model be specified with layers chosen such that their traveltimes all have the same constant value. Such models are generally called Goupillaud models, and the traveltime constant is usually taken to be the same as the time sample size of the seismogram to be computed. This is essentially a bookkeeping method that uses the diagram in Figure 4.6 to keep track of all waves. The

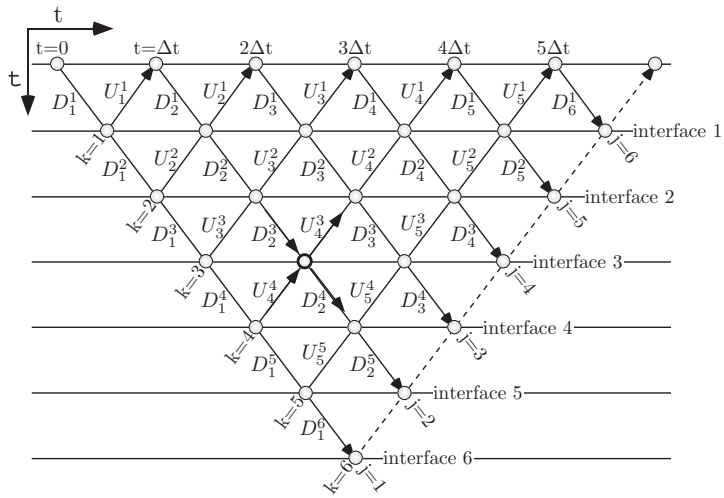


Figure 4.6 The computation triangle for a six-layer synthetic seismogram based on the Goupillaud model. Computation begins with a unit impulse in the upper left corner. Horizontal lines are interfaces between layers, diagonals slanting down from left to right are downgoing waves, and diagonals slanting up from left to right are upgoing waves. Nodes are denoted by their interface number, n, and their upgoing wave index, k. The highlighted node is $(n, k) = (3, 4)$.

vertical axis of this diagram is the model traveltime, τ, while the horizontal axis is the event arrival time, t. Layer boundaries are indicated by horizontal lines and indexed by the integer $n = 0, 1, 2, \ldots, N$, with 0 denoting the recording surface. As in the previous section, it is assumed that the model has been sampled such that the layers all have the same interval traveltime, $\tau_n - \tau_{n-1}$, and hence are of variable thickness in z. The diagonal lines slanting down to the right correspond to downward-traveling waves that are indexed by $j = 1, 2, 3, \ldots$ Similarly, the diagonals that slant up to the right denote upward-traveling waves using the index $k = 1, 2, 3, \ldots$ Though these waves are symbolized by diagonal lines, this is a 1D model, so all wave motion is actually vertical. The intersection of a downward wave and an upward wave occurs at a node that corresponds to a *scattering event*. A node is designated by the pair (n, k) that denotes its interface number and the upward-traveling wave index. The downward wave index at node (n, k) is $j = k - n + 1$.

The algorithm description requires that the upward- and downward-traveling waves be described by both their wave indices and the layer number. The layer number is the same as the interface number for the layer bottom. Thus the upward-traveling wave in layer n with wave index k will be denoted U_k^n, while the jth downward wave in the same layer will be denoted D_j^n. The fourth upward wave in the first layer is U_4^1, and so on. Considering a particular node (n, k), there are two incident waves, U_k^{n+1} and D_{k-n+1}^n, and two scattered waves, U_k^n and D_{k-n+1}^{n+1}. For example, the second node in the third layer in Figure 4.6 is node $(3, 4)$. The incident waves enter the node from the left and are U_4^4 and D_2^3, while the scattered waves exit the node from the right and are U_4^3 and D_2^4. For interface n, denoting the reflection coefficient for incidence from above by R_n, the scattered waves of *displacement* at node (n, k) are related to the incident waves by the equations

$$\begin{aligned} U_k^n &= [1 + R_n]\, U_k^{n+1} - R_n D_{k-n+1}^n, \\ D_{k-n+1}^{n+1} &= [1 - R_n]\, D_{k-n+1}^n + R_n U_k^{n+1}. \end{aligned} \tag{4.79}$$

For pressure waves, the corresponding equations are

$$\begin{aligned} Up_k^n &= [1 + R_n]\, Up_k^{n+1} + R_n Dp_{k-n+1}^n, \\ Dp_{k-n+1}^{n+1} &= [1 - R_n]\, Dp_{k-n+1}^n - R_n Up_k^{n+1}, \end{aligned} \tag{4.80}$$

where Up_k^n is the kth upward-traveling pressure wave in the nth layer and Dp_k^n is a similar downward-traveling pressure wave.

The displacement seismogram algorithm begins by assuming $D_1^1 = 1$, as is appropriate for an impulse response. At node $(1, 1)$ the upward incident wave, U_1^2, is zero and so the scattered waves are $D_1^2 = [1 - R_1]\, D_1^1 = 1 - R_1$ and $U_1^1 = -R_1$. In general, $U_k^n = 0$ if $n < k$ or $n < 1$ and $D_j^n = 0$ if $n < 1$ or $j < 1$. Then U_1^1 can be propagated up to the surface, where the scattered wave $D_2^1 = R_0 U_1^1$, a *surface-related multiple*, is calculated. Since all of the layers have the same two-way traveltime, Δt, the waves arrive at interface 0 at regular intervals, where they are accumulated into the seismogram. If surface-related multiples are to be included, the kth sample of the seismogram is $U_k^1 + D_{k+1}^1 = [1 + R_0]\, U_k^1$, otherwise it is just U_k^1.

Once U_1^1 has been propagated to the surface and D_1^2 and D_2^1 have been calculated, the upward wave U_2^2 can be calculated and propagated to the surface. U_2^2 is easy and is just $U_2^2 = -R_2 D_1^2 = -R_2[1 - R_1]$. Then, at node $(1, 2)$, D_2^1 and U_2^2 are incident and the scattered waves are $U_2^1 = [1 + R_1]\, U_2^2 - R_1 D_2^1$ and $D_2^2 = [1 - R_1]\, D_2^1 + R_1 U_2^2$. The last step in bringing U_2 to the surface is to calculate $D_3^1 = R_0 U_2^1$. Thus the second sample on the seismogram is determined and the downgoing waves D_1^3, D_2^2, and D_3^1 are all computed in preparation for propagation of U_3^3 to the surface.

The pattern should now be clear. The wave $U_k^k = -R_k D_1^k$ is propagated to the surface using Eqs. (4.79) to include the contributions from all of the downgoing multiples and to calculate the upgoing and downgoing scattered waves. The propagation of U_k^k to the surface results in the calculation of all of the downgoing waves that are required for U_{k+1}^{k+1}. The process then continues for as long as is desired. In principle, there are infinitely many arrivals from an N-layer system because there can be infinitely many multiple bounces.

Using z-transform notation, the first few samples of the displacement seismogram can be shown to be

$$\begin{aligned} u_{\text{imp}}(z) = 1 &- [1 + R_0] R_1 z^{\Delta t} - [1 + R_0] \Big[R_2 (1 - R_1^2) - R_1^2 R_0 \Big] z^{2\,\Delta t} \\ &- [1 + R_0] \Big[R_3 (1 - R_2^2)(1 - R_1^2) \\ &- R_2 R_1 (R_2 + R_0)(1 - R_1^2) - R_2 R_1 R_0 (1 - R_1^2) + R_1^3 R_0^2 \Big] z^{3\,\Delta t} + \ldots . \end{aligned} \tag{4.81}$$

The equivalent pressure seismogram is given by

$$\begin{aligned} p_{\text{imp}}(z) = 1 &+ [1 - R_0] R_1 z^{\Delta t} + [1 - R_0] \Big[R_2 (1 - R_1^2) - R_1^2 R_0 \Big] z^{2\,\Delta t} \\ &+ [1 - R_0] \Big[R_3 (1 - R_2^2)(1 - R_1^2) \\ &- R_2 R_1 (R_2 + R_0)(1 - R_1^2) - R_2 R_1 R_0 (1 - R_1^2) + R_1^3 R_0^2 \Big] z^{3\,\Delta t} + \ldots . \end{aligned} \tag{4.82}$$

The displacement and pressure seismograms have different physical units, which are not reflected in Eqs. (4.81) and (4.82).

The increasing complexity of each term is apparent even with only four samples written explicitly. In each term, the primary arrival can be identified by comparison with Eq. (4.76). In general, there are many possible multiples that also contribute to each term. In Eq. (4.81), each term after the leading one contains a common factor $1 + R_0^2$ that includes the surface-related multiple. In Eq. (4.82), the surface-related multiple is included via the term $1 - R_0^2$. If the surface is modeled as a *free surface*, then it has $R_0 = 1$ (recall that these reflection coefficients are defined for incidence from above), so each term in the displacement seismogram is doubled, while the pressure seismogram is identically zero. This is correct and consistent with the boundary condition of a liquid free surface that calls for the pressure to vanish.

Though the pressure seismogram of Eq. (4.82) vanishes identically on a free surface, the expression can be used to draw a valuable observation in comparison with the displacement seismogram of Eq. (4.81). If the two seismograms are added, then the signs are such that

the primaries cancel and only the surface-related multiples remain. Or, more importantly, the seismograms can be subtracted to cancel all of the surface-related multiples. Of course, this observation overlooks the fact that the pressure and displacement seismograms have different physical dimensions. In practice, this could be accounted for by scaling the first arrivals to have the same magnitude. These details aside, the important conclusion is that downgoing waves, like the surface-related multiple, can be eliminated by subtracting a scaled pressure seismogram from a displacement seismogram. This observation is the basis of the modern technique of marine seismic surveying with a cable laid on the ocean bottom. Called *OBC recording*, this method includes a paired hydrophone and geophone at each recording position.

More specifically, it is of interest to detail explicitly a few downgoing and upgoing waves for both displacement and pressure. For example, the downgoing waves needed to propagate U_3^3 to the surface are

$$\begin{aligned} D_1^3 &= (1 - R_2)(1 - R_1), \\ D_2^2 &= -R_1(1 - R_1)(R_0 + R_2), \\ D_1^3 &= -R_2R_1(1 - R_1^2) + R_1^2R_0^2, \end{aligned} \tag{4.83}$$

and the corresponding pressure waves, identical in form, are

$$\begin{aligned} Dp_1^3 &= (1 - R_2)(1 - R_1), \\ Dp_2^2 &= -R_1(1 - R_1)(R_0 + R_2), \\ Dp_1^3 &= -R_2R_1(1 - R_1^2) + R_1^2R_0^2. \end{aligned} \tag{4.84}$$

Then, propagation of the third upgoing wave to the surface gives, for displacement,

$$\begin{aligned} U_3^3 &= -R_3(1 - R_2)(1 - R_1), \\ U_3^2 &= -R_3(1 - R_2^2)(1 - R_1) + R_2R_1(1 - R_1)(R_0 + R_2), \\ U_3^1 &= -R_3(1 - R_2^2)(1 - R_1^2) + R_2R_1(1 - R_1^2)(R_0 + R_2) + R_2R_1R_0(1 - R_1^2) - R_1^3R_0^2 \end{aligned} \tag{4.85}$$

and, for pressure,

$$\begin{aligned} Up_3^3 &= R_3(1 - R_2)(1 - R_1), \\ Up_3^2 &= R_3(1 - R_2^2)(1 - R_1) - R_2R_1(1 - R_1)(R_0 + R_2), \\ Up_3^1 &= R_3(1 - R_2^2)(1 - R_1^2) - R_2R_1(1 - R_1^2)(R_0 + R_2) - R_2R_1R_0(1 - R_1^2) + R_1^3R_0^2. \end{aligned} \tag{4.86}$$

A term-by-term comparison of these upgoing wave expressions shows that a particular upgoing displacement wave is opposite in sign for every term when compared with the corresponding pressure wave. Though these calculations can be complex, the sign differences are a simple consequence of the scattering equations (4.79) and (4.80) and of the fact that the source has been placed above the entire model. Thus, if the initial downgoing displacement and pressure pulses are in phase, then the only upgoing waves are generated by reflections and these have opposite polarity for pressure versus displacement.

OBC recording can be modeled using the one-dimensional seismogram theory by calculating the recorded wave at the particular interface representing the ocean bottom. There, the pressure wave will not trivially vanish and the upgoing waves will have the polarity relationships just described. This can be done with the present algorithm by representing the water layer as many thin layers of equal traveltime and no impedance contrast. However, for a deep water layer and a typical seismogram sample rate of 0.001 s, this can mean a great many layers that serve no purpose other than to give the correct traveltime. In a later section, a synthetic-seismogram algorithm will be given that allows the layers to have arbitrary thicknesses. Then it will be possible to model a water layer with a single algorithmic layer.

As a final comment, this theory makes clear that the effect of multiple reflections is generally *nonstationary*. This means that the multiple content of the seismogram generally increases with time. Ordinary deconvolution theory, which assumes a stationary seismogram, cannot readily deal with a nonstationary multiple sequence. However, there are certain classes of multiples that have the desired stationarity. For example, in Eq. (4.81), the effect of the free surface is to cause every term after the direct arrival to have the same $1 + R_0$ multiplicative factor. This means that the free-surface multiples can be modeled as a convolution and that deconvolution might be expected to remove them. A similar effect is true for a bright reflector that occurs deeper in a model. All reflections from beneath the bright reflector will have the same characteristic multiple, while those from above it will not.

Exercises

4.6.2 On a replica of Figure 4.6, sketch the raypath for each multiple that arrives at $t = 3\,\Delta t$. How many multiples are there in total?

4.6.3 Verify the expression given for U_3^1 in Eq. (4.81). Also, compute the downgoing waves D_1^4, D_2^3, D_3^2, and D_4^1 in preparation for the next exercise. Verify the downgoing waves to be

$$
\begin{aligned}
D_1^4 &= (1 - R_3)(1 - R_2)(1 - R_1), \\
D_2^3 &= -R_1(1 - R_2)(1 - R_1)(R_0 + R_2) - R_3R_2(1 - R_2)(1 - R_1), \\
D_3^2 &= -R_2R_0(1 - R_1)(1 - R_1^2) + R_1^2R_0^2(1 - R_1) - R_3R_1(1 - R_2^2)(1 - R_1) \\
&\quad + R_2R_1^2(1 - R_1)(R_0 + R_2), \\
D_4^1 &= -R_3R_0(1 - R_2^2)(1 - R_1^2) + R_2R_1R_0(1 - R_1^2)(R_0 + R_2) + R_2R_1R_0^2(1 - R_1^2) \\
&\quad - R_1^3R_0^3.
\end{aligned}
$$

4.6.4 Calculate U_4^1 using the method described above with Figure 4.6 and the results of the previous exercise. This computation is algebraically tedious. The point of the exercise is to improve understanding of the complexity of the seismogram.

The answer is

$$\begin{aligned} U_4^1 = &- R_4(1-R_3^2)(1-R_2^2)(1-R_1^2) \\ &+ (1-R_2^2)(1-R_1^2)\left[2R_3R_1R_0 + 2R_3R_2R_1 + R_3^2R_2\right] \\ &+ (1-R_1^2)\left[R_2^2R_0(1-R_1^2) - 3R_2R_1^2R_0^2 - 2R_2^2R_1^2R_0 - R_2^3R_1^2\right] + R_1^4R_0^3. \end{aligned}$$

4.7 MATLAB Tools for 1D Synthetic Seismograms

There are a number of different possibilities for creating synthetic seismograms in MATLAB. The intended purpose of the synthetic usually dictates which facility to use. For example, the testing of spiking deconvolution algorithms is usually done with a random reflectivity, no multiples, and a minimum-phase wavelet, while testing predictive deconvolution will usually involve the inclusion of some form of multiples. Alternatively, a synthetic seismogram intended for comparison with processed seismic data will usually be created using a well-log reflectivity, no multiples, and a zero-phase wavelet.

4.7.1 Wavelet Utilities

Once an impulse-response seismogram has been created, it is usually convolved with a wavelet to simulate the band limiting imposed by the seismic source. There are a number of different wavelets available, including:

`ormsby` Creates an Ormsby band-pass filter.
`ricker` Creates a Ricker wavelet.
`sweep` Generate a linear Vibroseis sweep.
`wavemin` Creates a minimum-phase wavelet for impulsive sources.
`wavevib` Creates a Vibroseis (Klauder) wavelet.
`wavez` Creates a zero-phase wavelet with a dominant frequency.

These functions all have a similar interface, in that they accept a number of input parameters describing the construction of the wavelet and return only two outputs: the wavelet and its time-coordinate vector. Since only `wavemin` and `sweep` generate causal responses, the time-coordinate vector is essential to properly position the wavelet.

The `ormsby` command, used in Code Snippet 4.7.1, produces a popular zero-phase wavelet that has an easy specification of its passband. The analytic expression for the Ormsby wavelet is

$$\begin{aligned} w(t)_{\text{ormsby}} = &\frac{\pi f_4^2}{f_4 - f_3}\left[\frac{\sin \pi f_4 t}{\pi f_4 t}\right]^2 - \frac{\pi f_3^2}{f_4 - f_3}\left[\frac{\sin \pi f_3 t}{\pi f_3 t}\right]^2 \\ &- \frac{\pi f_2^2}{f_2 - f_1}\left[\frac{\sin \pi f_2 t}{\pi f_2 t}\right]^2 + \frac{\pi f_1^2}{f_2 - f_1}\left[\frac{\sin \pi f_1 t}{\pi f_1 t}\right]^2, \end{aligned} \tag{4.87}$$

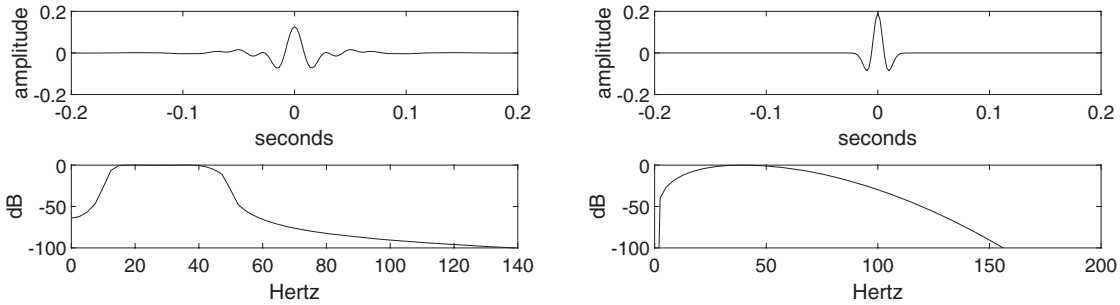


Figure 4.7a (left) An Ormsby wavelet for the passband 10, 15, 40, 50 Hz is shown in the time domain (top) and its amplitude spectrum is shown against a decibel scale in the bottom frame. This figure was created by Code Snippet 4.7.1.

Figure 4.7b (right) A Ricker wavelet with a 40 Hz dominant frequency is shown in the time domain (top) and its amplitude spectrum on a decibel scale is shown in the bottom frame. This was created by Code Snippet 4.7.2.

Code Snippet 4.7.1 This code creates the Ormsby wavelet shown in Figure 4.7a. The inputs to *ormsby* are the four frequencies defining the passband, the wavelet length, and the sample rate.

```
1  %make ormsby wavelet
2  [w,tw]=ormsby(10,15,40,50,.4,.002);
3  %compute spectrum
4  [W,f]=fftrl(w,tw);
5  W=todb(W);
```

End Code

wavepropcode/ormsby_example.m

Code Snippet 4.7.2 This code creates the Ricker wavelet shown in Figure 4.7b. The inputs to *ricker* are the temporal sample rate (in seconds), the dominant frequency (in hertz), and the temporal length (in seconds).

```
1  %make ricker wavelet
2  [w,tw]=ricker(.002,40,.4);
3  %compute spectrum
4  [W,f]=fftrl(w,tw);
5  W=todb(W);
```

End Code

wavepropcode/ricker_example.m

where $f_1 < f_2 < f_3 < f_4$ are the specifications of the frequency passband in hertz. Approximately, the passband begins at f_1 and increases to full amplitude at f_2. It remains at full amplitude until f_3 and ramps down to the stop band at f_4. Figure 4.7a shows an Ormsby wavelet with $(f_1, f_2, f_3, f_4) = (10, 15, 40, 50)$ in the time domain and its Fourier amplitude

spectrum. On line 2 in Code Snippet 4.7.1, the wavelet passband is specified as the first four input parameters, the fifth parameter is the desired wavelet length in seconds, and the sixth parameter is the temporal sample rate, also in seconds. It is obviously important to ensure that the Nyquist frequency, $f_{\text{nyq}} = 1/(2\,\Delta t)$, is greater than f_4. Less obviously, the temporal length of the wavelet must be sufficient to ensure that the character of the wavelet is captured. Since the frequency bandwidth and the temporal width are inversely proportional, a narrow-passband wavelet will be temporally long, and vice versa. With any of these wavelet tools, it is always a good idea to plot the wavelet before using it and assess the adequacy of the temporal length. If the length is chosen too short, then truncation effects will distort the desired spectrum. Line 4 of Code Snippet 4.7.1 computes the wavelet's spectrum using `fftrl`, and line 5 computes the decibel amplitude spectrum using the utility `todb`. Plotting the real part of the output from `todb` versus `f` creates the spectrum shown in Figure 4.7a.

An important consideration when creating wavelets is the question of normalization. That is, how should the overall amplitude of the wavelet be determined? If Eq. (4.87) is used directly, the maximum amplitude of the wavelet can be quite large. This may be an annoyance in that any time series that is convolved with such a wavelet will have a large change in overall amplitude. An easy solution might seem to be to scale the wavelet such that its maximum absolute value is unity. However, another possibility is to scale the wavelet such that a sinusoid of the dominant frequency is passed at unit amplitude under convolution. These criteria are not identical, with the first tending to preserve the peak values of the signal before and after convolution and the second to preserve the amplitude of the signal near the wavelet's dominant frequency. The wavelets presented here all use the second method of normalizing with respect to the dominant frequency. However, if other behavior is desired, the wavelet can always be renormalized after it is created. The function `wavenorm` is useful for this purpose.

Though the amplitude spectrum of the Ormsby wavelet in Figure 4.7a shows the desired passband, the wavelet is often felt to have an unacceptable level of ripple. The Ricker wavelet has a simpler form in the time domain, though this comes at the expense of a broader, less controlled, passband. The Ricker wavelet is given analytically by

$$w(t)_{\text{ricker}} = \left[1 - 2\pi^2 f_{\text{dom}}^2 t^2\right] e^{-\pi^2 f_{\text{dom}}^2 t^2}, \tag{4.88}$$

which can be shown to be the second derivative of a Gaussian. In this expression, f_{dom} is the *dominant frequency* of the wavelet. The Fourier transform of this wavelet is known to be (Sheriff and Geldart, 1995)

$$W(f) = \frac{2f^2}{\sqrt{\pi} f_{\text{dom}}^2} e^{-f^2/f_{\text{dom}}^2}. \tag{4.89}$$

This is a Gaussian multiplied by f^2/f_{dom}^2. It is characteristic of Ricker wavelets that the higher the dominant frequency, the broader the bandwidth. This is evident from Eq. (4.89) in that the effective width of the Gaussian is seen to be f_{dom}. The effect is demonstrated by Code Snippet 4.7.3 and shown in Figure 4.8a.

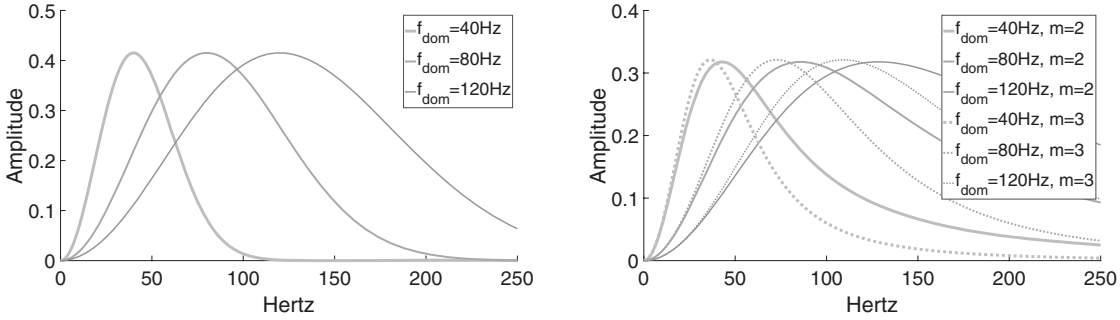


Figure 4.8a (left) Three Ricker wavelet spectra are shown against a linear scale for $f_{\text{dom}} = 40$, 80, and 120 Hz. This figure was created by Code Snippet 4.7.3 using Eq. (4.89).

Figure 4.8b (right) Six amplitude spectra are shown illustrating the behavior of Eq. (4.90). Three dominant frequencies are shown, $f_{\text{dom}} = 40$, 80, 120, and the bold curves correspond to $m = 2$, while the dashed curves are for $m = 3$.

Code Snippet 4.7.3 This code calculates the Ricker wavelet spectra shown in Figure 4.8a using Eq. (4.89).

```
f=0:250;
fdom1=40;fdom2=80;fdom3=120;
R1=exp(-f.^2/(fdom1^2)).*(2*f.^2)/(sqrt(pi)*fdom1^2);
R2=exp(-f.^2/(fdom2^2)).*(2*f.^2)/(sqrt(pi)*fdom2^2);
R3=exp(-f.^2/(fdom3^2)).*(2*f.^2)/(sqrt(pi)*fdom3^2);
```

End Code

wavepropcode/rick_spec_ex.m

The Ormsby and Ricker wavelets are zero phase (i.e., symmetric) and are most suitable for models to be compared with fully processed seismic images. An alternative is minimum-phase wavelets, which are suitable for models to be compared with raw seismic data or to be used for deconvolution testing. One effective way to design minimum-phase wavelets is to specify the locations of the poles and zeros of the wavelet's z-transform directly. This is the approach taken by Claerbout (1976) and has the advantage that the minimum-phase property is easily assured. On the other hand, it is not necessarily easy to control the spectral shape. With this in mind, an alternative is presented here that specifies the wavelet's amplitude spectrum and calculates the minimum-phase wavelet using the *Levinson recursion*. The Levinson recursion is an efficient way to solve a system of linear equations that has *Toeplitz* symmetry. It is discussed more completely in the conjunction with Wiener deconvolution in Chapter 5. For now, it is just presented as an abstract numerical technique.

Since a minimum-phase wavelet is meant to model an impulsive source, an analytic model for the amplitude spectrum should resemble the observed spectrum for the dynamite blast that is common in exploration seismology. Such spectra have a dominant frequency below about 30 Hz, with larger charges giving lower dominant frequencies, and show slow

Code Snippet 4.7.4 This code is an excerpt from *wavemin* and illustrates the calculation of a minimum-phase wavelet. The column vector of frequencies is `f`, the intended length of the wavelet is `tlength`, the dominant frequency is `fdom`, and the temporal sample rate is `dt`.

```
1   % create the power spectrum
2     powspec=tntamp(fdom,f,m).^2;
3   % create the autocorrelation
4     auto=ifftrl(powspec,f);
5   % run this through Levinson
6     nlags=tlength/dt+1;
7     b=[1.0 zeros(1,nlags-1)]';
8     winv=levrec(auto(1:nlags),b);
9   % invert the wavelet
10    wavelet=real(ifft(1. ./(fft(winv))));
11    twave=(dt*(0:length(wavelet)-1)';
12  % now normalize the wavelet
13    wavelet=wavenorm(wavelet,twave,2);
```

End Code

wavepropcode/wavemin_ex.m

decay toward high frequencies. The model spectrum adopted here is given by

$$A(f) = \frac{1 - e^{-(f/f_{\text{dom}})^2}}{1 + (f/f_{\text{dom}})^m}, \tag{4.90}$$

where m determines the rate of decay at high frequencies. This spectrum is produced by *tntamp*. Figure 4.8b was created using *tntamp* and shows the behavior of this equation for $m = 2$ and $m = 3$. Clearly, larger m gives a more sharply band-limited spectrum, and this means a longer temporal wavelet. When using $m > 2$, care must be taken that a sufficient temporal length is selected, because truncation effects can destroy the minimum-phase property.

The function *wavemin* invokes *tntamp* to create a minimum-phase wavelet as shown in Code Snippet 4.7.4. Line 2 creates the desired power spectrum by squaring the output from *tntamp*. On line 4, the power spectrum is inverse Fourier transformed to create the autocorrelation of the desired wavelet. Lines 6–8 set up a linear system and solve it using the Levinson recursion. This linear system is derived in Chapter 5, where it is shown to specify the minimum-phase inverse of a minimum-phase wavelet. It is of the form $\underline{\underline{A}}\underline{x} = \underline{b}$, where $\underline{\underline{A}}$ is a Toeplitz matrix formed from the autocorrelation, $\underline{x}$ is the unknown minimum-phase inverse, and $\underline{b}$ is a column vector of zeros except for a one in the first place. Solving this system with *levrec* does not require the formation of the matrix $\underline{\underline{A}}$ but needs only the first n lags of the autocorrelation and the vector $\underline{b}$. Since the result from solving this linear system is the minimum-phase inverse, the desired wavelet is found by inverting the inverse. This is done in the Fourier domain on line 10. Finally, line 11 builds the time-coordinate vector and line 13 normalizes the final wavelet.

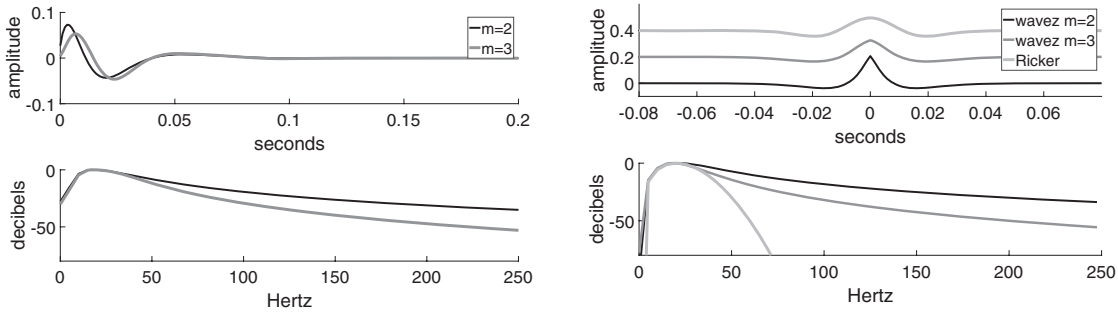


Figure 4.9a (left) Two minimum-phase wavelets (top) and their amplitude spectra (bottom). The wavelets were created with Code Snippet 4.7.5 and correspond to $m = 2$ and $m = 3$ in Eq. (4.90). The $m = 2$ wavelet has a broader spectrum and is more front-loaded in the time domain.

Figure 4.9b (right) In the upper frame, a 20 Hz Ricker wavelet (top) is compared with *wavez* wavelets corresponding to $m = 3$ (middle) and $m = 2$ (bottom). Their amplitude spectra are shown in the lower frame. The Ricker spectrum is the one that runs off the bottom of the axes at 70 Hz.

Code Snippet 4.7.5 This code calls *wavemin* twice, with $m = 2$ and $m = 3$, to create two different minimum-phase wavelets with the same dominant frequency. Their amplitude spectra are also calculated, and the result is shown in Figure 4.9a.

```
%make wavemin wavelet
[w1,tw]=wavemin(.001,20,.2,2);
[w2,tw]=wavemin(.001,20,.2,3);
%compute spectrum
[W1,f]=fftrl(w1,tw);
W1=todb(W1);
[W2,f]=fftrl(w2,tw);
W2=todb(W2);
```

End Code

wavepropcode/wavemin_example.m

The example shown in Code Snippet 4.7.5 uses *wavemin* to create two different minimum-phase wavelets. The wavelets have the same dominant frequency of 20 Hz and have $m = 2$ and $m = 3$. The wavelet with $m = 2$ shows less high-frequency decay and, in the time domain, is more *front loaded*. It is not certain that the result from *wavemin* will be truly minimum phase. If the parameters are chosen such that the requested temporal length is substantially shorter than what is implied by the spectrum, then a nonminimum-phase wavelet can result owing to truncation effects. The utility function *ismin* is provided to test a wavelet for the minimum-phase property. This function factors the z-transform of the wavelet and measures the radius vector for all of the roots. If all roots are outside the unit circle, then *ismin* returns unity. If any root is inside the unit circle, *ismin* returns 0. The two wavelets in Figure 4.9a pass the minimum-phase test; however, `wavemin(.001,10,.2,3)` produces a wavelet that does not.

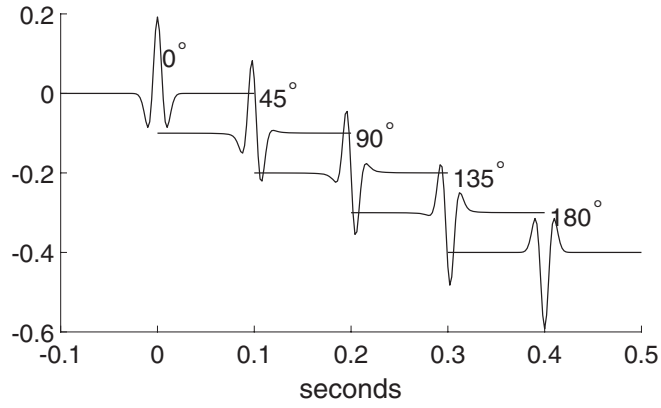


Figure 4.10 A 40 Hz zero-phase Ricker wavelet is shown for four phase rotations of $\theta = [45°, 90°, 135°, 180°]$. This was created with Code Snippet 4.7.6.

The spectral model of Eq. (4.90) can also be used to create a zero-phase wavelet. This is accomplished by `wavez`. Figure 4.9b shows three zero-phase wavelets with the same dominant frequency of 20 Hz. The upper wavelet is a 20 Hz Ricker wavelet, while the middle and lower were created by `wavez` with $m = 3$ and $m = 2$, respectively. Relative to the Ricker, the `wavez` wavelets are considerably more spikelike at the origin. This is due to their relative richness in high frequencies compared with the Ricker wavelet. The code that created this figure is a simple modification of Code Snippet 4.7.5 and is not shown.

Given any particular amplitude spectrum, there are infinitely many wavelets that can be created with different phase spectra. So far, only minimum phase and zero phase have been discussed. Often, it is useful to model seismic data with a wavelet that has a constant phase other than zero. This is easily accomplished by creating a zero-phase wavelet and then using `phsrot` to rotate the phase from zero to the desired constant value. An example of this, which creates Figure 4.10, is shown in Code Snippet 4.7.6. It is good practice to note the basic features of each phase rotation. For example, the 90° rotation has antisymmetry about the wavelet's center, with a major leading peak and a trailing trough. The 45° rotation is similar, but the major leading peak is about 50% larger than the trailing trough. Since $e^{i\pi} = -1$, a 180° phase rotation is just a flip in polarity. This can be seen in the 0° and 180° wavelets and it also means that a −45° rotation is the same as a polarity-reversed 135° wavelet. Similarly, the 45° wavelet is a polarity-reversed version of a −135° wavelet.

4.7.2 Convolutional Seismograms with Random Reflectivity

A common use for synthetic seismograms is to test deconvolution algorithms. In this context, it is often desirable to use an artificial reflectivity function so that the deconvolution assumption of a *white reflectivity* is fulfilled as nearly as possible. The command `reflec` supplies such a reflectivity function by using MATLAB's random number generator, `randn`. Technically, numerical random number generators such as `randn` generate

Code Snippet 4.7.6 This code creates a zero-phase Ricker wavelet and then calls *phsrot* four times to create a set of phase-rotated wavelets. The resulting plot is shown in Figure 4.10.

```
[w,tw]=ricker(.002,40,.2);%make Ricker wavelet
nrot=5;deltheta=45;%number of phases and increment
figure
for k=1:nrot
    theta=(k-1)*deltheta;
    wrot=phsrot(w,theta);%phase rotated wavelet
    xnot=.1*(k-1);ynot=-.1*(k-1);
    line(tw+xnot,wrot+ynot,'color','k');%plot each wavelet
    text(xnot+.005,ynot+.1,[int2str(theta) '^\circ'])
end
```

End Code

wavepropcode/phsrot_example.m

pseudo-random numbers. This means that, given the same starting point, called the *seed*, the generator will return the same set of apparently random numbers. MATLAB has two random number generators, `rand`, which generates *uniformly distributed* random numbers, and `randn`, which generates *normally*, or *Gaussian*, *distributed* random numbers. Generally, a normal distribution is preferable; however, in either case, such random numbers do not display the *spikiness* typically found in real well logs. This is addressed in *reflec* by raising the output from `randn` to an integral power, m, as in

$$r_{\text{reflec}} = \text{sign}(r_{\text{randn}})\,|r_{\text{randn}}|^m\,. \tag{4.91}$$

Suppose that the maximum values in r_{randn} are near unity. Then raising them to any power will not change them, while the smaller numbers will get smaller. Thus r_{reflec} will appear more spiky than r_{randn}.

Figure 4.11a compares a real reflectivity derived from well logs with six outputs from *reflec* using values of m from 1 to 6. It is a subjective decision which value of m gives the most realistic reflectivity estimate. The default in *reflec* is 3. Clearly, the real reflectivity has features that are present in none of the synthetic ones. The most obvious is the low-amplitude zone from 0.4 to 0.7 s in the real reflectivity. More generally, a low-frequency amplitude variation can be perceived throughout the real reflectivity, while the synthetic ones show a very uniform amplitude distribution with time. This is a hint that the real reflectivity is not white, i.e., it is not truly random, but rather shows *spectral color*.

A useful test of randomness is to calculate the autocorrelation of the reflectivity. In theory, a truly random signal has an autocorrelation that is exactly zero everywhere except at zero lag, where it is unity. However, this requires an infinite-length signal and so is never exactly fulfilled in practice. Figure 4.11b shows the central portion of the autocorrelations of the reflectivities of Figure 4.11a. All autocorrelations have a large spike at zero lag and only small values elsewhere. However, only the real well log shows significant negative sidelobes of the zero-lag sample. This again indicates that the real reflectivity is not white

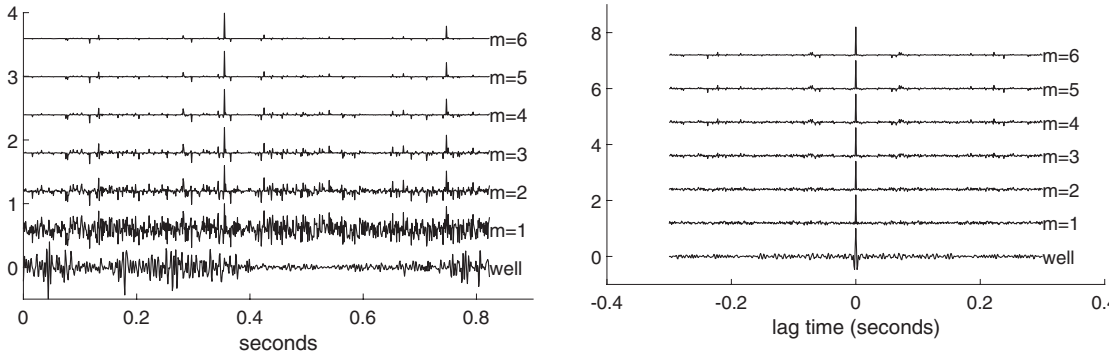


Figure 4.11a (left) Seven reflectivity functions. The bottom one comes from a real well log, and the other six are synthetic reflectivities that were generated by `reflec`.

Figure 4.11b (right) Seven two-sided autocorrelations, corresponding to the seven reflectivities in Figure 4.11a.

Code Snippet 4.7.7 Here two wavelets, a 40 Hz Ricker and a 20 Hz minimum-phase wavelet, are convolved with the same reflectivity to generate two synthetic seismograms. The result is shown in Figure 4.12a.

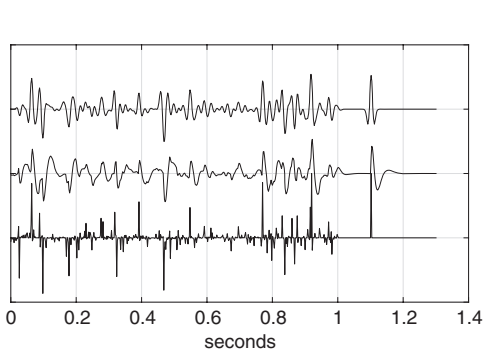
```
1   %make wavelet
2   dt=.002;tmax=1;tmaxw=.2;m=2;
3   [wm,twm]=wavemin(dt,20,tmaxw,m); %20 Hz min phs
4   [wr,twr]=ricker(dt,40,tmaxw); %40 Hz Ricker
5   %make reflectivity
6   [r,t]=reflec(tmax,dt,.2);
7   %pad spike at end
8   ntw=tmaxw/dt;
9   r=[r;zeros(ntw/2,1);.2;zeros(ntw,1)];
10  t=dt*(0:length(r)-1);
11  %convolve and balance
12  s1=convz(r,wr);
13  s1=balans(s1,r);
14  s2=convm(r,wm);
15  s2=balans(s2,r);
```

End Code

wavepropcode/ reflec_seis .m

but has spectral color. It is also interesting that r_{reflec} shows the same degree of randomness regardless of the value of m.

The calculation of a synthetic seismogram using `reflec` to generate the reflectivity is illustrated in Code Snippet 4.7.7. Lines 3 and 4 generate a 40 Hz Ricker wavelet and a 20 Hz minimum-phase wavelet as described in Section 4.7.1. Line 6 invokes `reflec` to create a random reflectivity. The three input parameters are the maximum time, the time sample rate, and the maximum reflection coefficient. A fourth argument, which is defaulted here, gives m in Eq. (4.91). In addition to returning a reflectivity series, `r`, `reflec` also



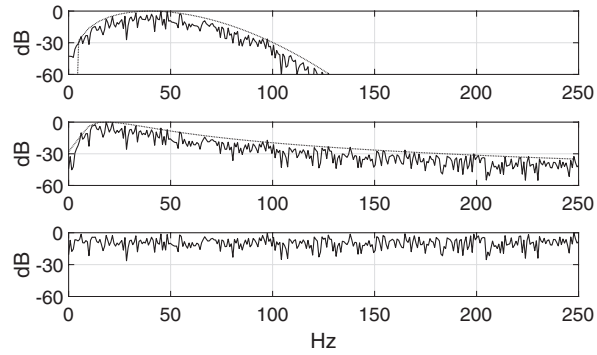


Figure 4.12a (left) The lower trace is a random reflectivity generated by `reflec` with an extra spike on the end. The middle trace is a convolutional seismogram that uses a 20 Hz minimum-phase wavelet convolved with the first trace. The top trace is a similar seismogram except that a 20 Hz minimum-phase wavelet was used. See Code Snippet 4.7.7.

Figure 4.12b (right) Decibel amplitude spectra of the seismograms of Figure 4.12a: bottom, reflectivity; middle, minimum-phase seismogram and wavelet (smooth curve); and top, Ricker seismogram and wavelet (smooth curve).

returns an appropriate time-coordinate vector, `t`. Lines 8–10 place an extra isolated spike on the end of the reflectivity. This is useful because, after convolution, the shape of the wavelet will be immediately apparent. Line 9 is simply the concatenation of the reflectivity with a column vector of `ntw/2` zeros, an impulse of 0.2, and another column vector of `ntw` zeros. Line 10 rebuilds the time-coordinate vector to be compatible with the lengthened reflectivity. Line 12 convolves the reflectivity with the zero-phase Ricker wavelet using the convenience function `convz`, while line 14 convolves the reflectivity with the minimum-phase wavelet using `convm`. These convenience functions invoke `conv` and then truncate the result to be the same length as the first input argument. The function `convz` truncates evenly from the beginning and the end of the convolved series, as is appropriate if a zero-phase wavelet has been used. The function `convm` truncates entirely off the end of the convolved series, as would be expected for a causal wavelet. After each convolution, the function `balans` is invoked to balance the amplitudes of the seismogram with the original reflectivity to aid in the comparison. One thing that is immediately apparent from these seismograms is that the zero-phase seismogram is more *interpretable* than the minimum-phase one. This is because the larger peaks on the former coincide temporally with the major reflection coefficients. With the minimum-phase seismogram, the major peaks are displaced to later times relative to the reflectivity.

Code Snippet 4.7.8 assumes that Code Snippet 4.7.7 has already been run. Lines 1–5 use `fftrl` to compute the spectrum of each component of the seismogram construction exercise. Then, lines 6–10 use the convenience function `todb` to compute the amplitude spectrum in decibel format, each spectrum relative to its own maximum. (The imaginary part of the return from `todb` is the phase spectrum.) The resulting spectra are displayed in Figure 4.12b. It is apparent that the spectra of the seismograms are shaped by those of the wavelets. That is, the slowly varying, average behavior of the seismogram spectrum

Code Snippet 4.7.8 This example assumes that Code Snippet 4.7.7 has already been run. Here the spectra are computed and displayed. The result is Figure 4.12b.

```
1   [R,f]=fftrl(r,t);
2   [S1,f]=fftrl(s1,t);
3   [S2,f]=fftrl(s2,t);
4   [Wr,fwr]=fftrl(wr,twr);
5   [Wm,fwm]=fftrl(wm,twm);
6   R=real(todb(R));
7   S1=real(todb(S1));
8   S2=real(todb(S2));
9   Wr=real(todb(Wr));
10  Wm=real(todb(Wm));
```

End Code

wavepropcode/ reflec_spec .m

is essentially similar to the wavelet spectrum. The fact that the Ricker wavelet spectrum departs from that of the seismogram in the upper panel is an artifact caused by the truncation of the convolution.

It is often desirable to add some level of background random noise to a synthetic to simulate a noisy recording environment or perhaps the limited precision of recording. The function *rnoise* is designed for this purpose. If `s1` is a time series, then `rn=rnoise(s1,5)` creates a vector of normally distributed random noise, `rn`, such that `s1+rn` will have a signal-to-noise ratio of 5. The signal-to-noise ratio is defined by assuming that `s1` is pure signal and measuring its rms amplitude as $a_{\text{rms}} = \sqrt{\sum_j s1_j^2}$, where the sum is over samples in the time domain. The rms amplitude of `rn` is then set as σa_{rms}, where σ is the desired signal-to-noise ratio. In using *rnoise* it is important to remember that `rn` must be added to the input time series after calling *rnoise*. Also, the method of calculating the signal-to-noise ratio may not be exactly what was desired, as there is no standard way of doing this. Other possibilities include defining the ratio between peak amplitudes or in the frequency domain. Also, *rnoise* can be given a third argument that defines the range of indices over which the signal-to-noise ratio is to be computed. This is useful for nonstationary signals whose signal level may change strongly with time. For example, if `t` is the time coordinate vector for `s1`, then `rn=rnoise(s1,5,near(t,.5,1.0))` uses the function *near* to determine the range of indices that span the time zone from 0.5 to 1.0 s. Then only those samples between 0.5 and 1 are used in the rms amplitude calculation. Figure 4.13a shows four different noisy seismograms computed using *rnoise* and the minimum-phase seismogram of Figure 4.12a as signal. The Fourier amplitude spectra of these noisy seismograms are shown in Figure 4.13b with the spectrum of the wavelet superimposed. Comparison with Figure 4.12b shows that the noise causes a *corner* in the spectrum, defined by where it departs from the shape of the wavelet's spectrum. An interpretive assessment of the corner frequencies would put them near 75, 100, 150, and 220 Hz for signal-to-noise ratios of 1, 2, 5, and 10, respectively. It is very difficult to recover the signal spectrum beyond this

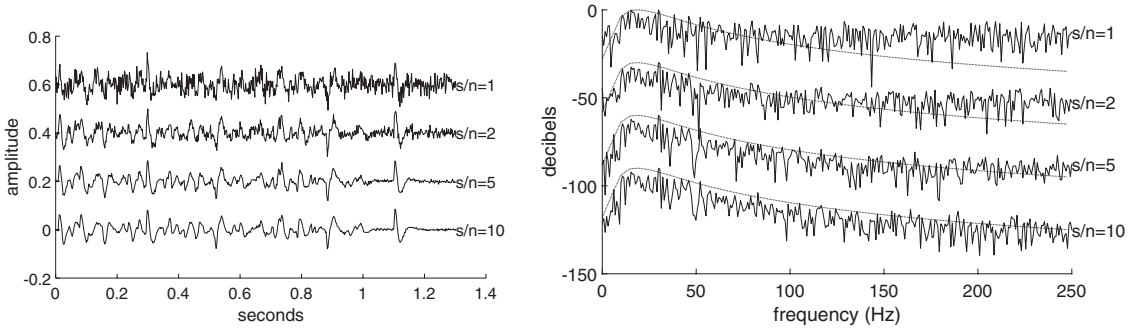


Figure 4.13a (left) The minimum-phase seismogram of Figure 4.12a is shown with four different levels of added noise.

Figure 4.13b (right) The spectra of the seismograms of Figure 4.13a are shown with the spectrum of the wavelet superimposed on each. The maxima of the spectra actually coincide but have been shifted vertically for clarity.

point in data processing. Close inspection shows that the spectra at frequencies below the corner frequency are also altered by the addition of noise. From another perspective, this must be the case because the dynamic range of the wiggle trace plots in Figure 4.13a is about 20–30 dB and they are all clearly altered by the noise.

Exercises

4.7.1 Develop a script that remakes and displays the seismograms and spectra of Figures 4.12a and 4.12b. Modify the script to demonstrate that truncation effects are the reason that the seismogram spectrum and the Ricker wavelet spectra depart from one another near 110 Hz (top of Figure 4.12b).

4.7.2 Create a script that builds two seismograms from the same reflectivity using wavelets from `wavemin` and `wavez`. Make the wavelets the same in all respects except for their phase. Which seismogram is more interpretable? Why?

4.7.3 Seismic Attenuation by the Constant-*Q* Theory

In Figure 4.13b, it is apparent that different signal-to-noise ratios lead directly to different signal bandwidths. However, since the signal-to-noise ratio is defined in the time domain and the signal bandwidth is a frequency-domain measure, it is not a simple matter to relate these concepts. Moreover, in all real seismic data, the wavelet attenuates as it propagates owing to *anelastic* (or *viscoelastic*) loss. This term refers to a general process whereby wave energy is lost irreversibly to heat as the wave propagates. There are many suggested models for this process but no general agreement on its physical detail. The empirical fact is that as a seismic wave propagates, its amplitude spectrum shows progressive loss of high frequencies and the phase spectrum undergoes progressive phase rotations. This energy loss is commonly parameterized by the quantity Q, or the *quality factor* of rocks,

defined as the energy/(energy loss) per wave cycle. Given this definition, Q approaches ∞ for perfectly elastic materials (i.e., no energy loss), while values ranging between 10 and 200 are commonly reported in field measurements. Low Q values correspond to lossy material such as soil or poorly compacted rocks, while values above 100 are usually only found in very rigid rocks. A theoretical argument found in Udias (1999) (p. 260) suggests that Q for shear waves will be about half of that for pressure waves.

The effects of anelastic attenuation are present in all seismic data and range from subtle to dramatic. Despite the guaranteed presence of these effects, they are often neglected in seismic modeling and processing, perhaps because the theoretical details can be challenging (e.g., Borcherdt (2009)) and the effects are often subtle. Also, underlying all models of viscoelasticity there is an ill-defined and nonunique empirical model of rock behavior that describes the actual loss mechanism. However, the tools discussed here are based on the constant-Q model (Kjartansson, 1979), which is a widely accepted approximation to observed attenuation behavior. The constant-Q model refers to a Q that is independent of frequency but may still be a function of position. This model is mathematically simple and provides a good description of the first-order effects of attenuation. Among these first-order effects are (1) time- and frequency-dependent attenuation, (2) minimum-phase dispersion, (3) measurable traveltime differences between sonic logging frequencies and seismic frequencies (commonly called *drift*), and (4) frequency-dependent reflectivity. The various functions described here provide relatively simple access to these effects and allow them to be studied in relation to other seismic processes such as deconvolution.

Kjartansson (1979) gives a full mathematical description of constant-Q wave propagation for scalar waves, while Borcherdt (2009) gives a complete discussion of viscoelastic wave propagation. Both of these works are beyond the scope of the present discussion; however, a relatively simple description can be built on Kjartansson's formula for the Fourier transform of the impulse response of a 1D constant-Q material, which will be called the Q *wavelet*. This formula is

$$\widehat{w_Q}(f,x) = A(f,x)e^{2\pi ifx/v(f)}, \tag{4.92}$$

where x is the distance traveled, f is the frequency (always positive), $A(f)$ is the amplitude spectrum, given by

$$A(f,x) = e^{-\pi xf/(Qv_0)}, \tag{4.93}$$

and $v(f)$ is the frequency-dependent velocity

$$v(f) = v_0\left(1 - \frac{1}{\pi Q}\ln\frac{f}{f_0}\right)^{-1}, \tag{4.94}$$

where v_0 is the velocity at frequency f_0. Physically, Eq. (4.92) predicts the evolution of an initial impulse as it travels through a 1D attenuating medium (see Figure 4.16a later in this section). These expressions emerge from Kjartansson's theory, which is not presented here, but for which there is strong empirical evidence. Instead, we will focus on understanding the implications of the theory.

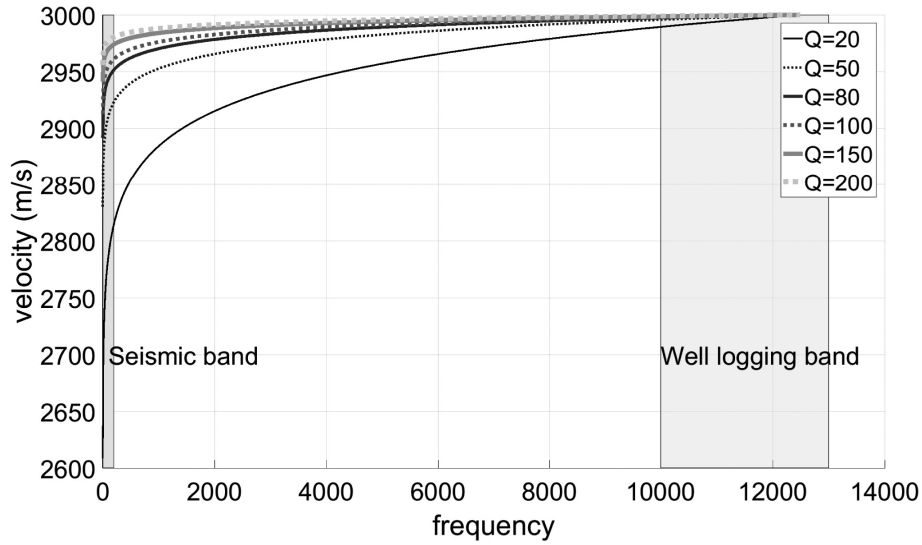


Figure 4.14 The frequency dependence of velocity, as predicted by constant-Q theory, is illustrated for a velocity of 3000 m/s measured at a frequency of 12 500 Hz. This simulates a well-logging measurement and illustrates that such measurements are generally faster than similar measurements made from seismic reflection data. The shaded gray areas indicate the frequency range of reflection seismic data (0–200 Hz) and well-logging data (10 000–13 000 Hz). This is predicted by Eq. (4.94).

First, consider the expression for the amplitude spectrum (Eq. (4.93)) and replace $t = x/v_0$ to get

$$A(f,t) = e^{-\pi ft/Q}. \tag{4.95}$$

This is the fundamental prediction for the time-variant amplitude spectrum as predicted by constant-Q theory. It says that the spectrum will decay exponentially with both time and frequency. At $t = 0$, $A(f,t) = 1$, which indicates that this theory assumes an initial delta-function impulse. $A(f,t)$ is constant along curves defined by $ft =$ constant, which are hyperbolas in the time–frequency domain (assuming Q to be a constant). The implication is that if we observe a maximum signal frequency of f_1 at time t_1, then at some later time t_2 we expect the maximum signal frequency to have decreased to $f_2 = f_1 t_1/t_2 < f_1$.

Next, consider the frequency-dependent velocity of Eq. (4.94). Figure 4.14 illustrates the behavior of Eq. (4.94) for several different values of Q. Generally, well-logging velocity measurements are conducted at very high frequencies, often above 10 000 Hz, while the seismic reflection frequency band is usually below 200 Hz. As shown here, the corresponding measurement at seismic frequencies can be considerably lower. This means that a synthetic seismogram constructed directly from well-log velocities will show events at earlier times than those seen in real data. This effect is cumulative, so, compared with a synthetic seismogram computed from uncorrected velocities, a real seismic trace will show events with a continually increasing lag toward greater times, as is illustrated in Figure 4.15. The calculation of traveltime through a $v(z)$ medium, as described by a well

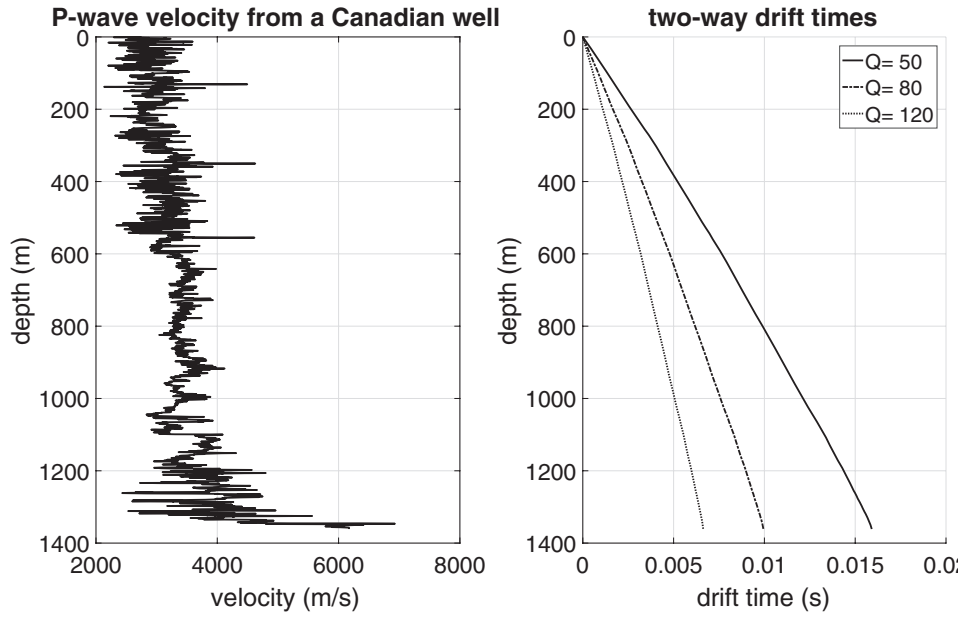


Figure 4.15 (left) P-wave velocity measurements from a well in the Western Canadian sedimentary basin. Logging started at 200 m depth and extended to over 1200 m. (right) The drift times as expected from these velocity measurements for three different Q values. The logging frequency was taken to be 12 500 Hz and the seismic frequency was assumed to be 30 Hz. The computation required for this figure is in Code Snippet 4.7.9.

log, is a simple matter of summing the discrete traveltimes for each implied layer. Given a list of velocities and depths like $v_1, v_2, \ldots, v_n$, and $z_1, z_2, \ldots, z_n$, then there are two possible layered media, implied by the choice to assign v_k to the interval $[z_{k-1}, z_k]$ or the interval $[z_k, z_{k+1}]$. Choosing the first notation and accepting that generally z_1 will be greater than 0 (logging right to the surface is rare and difficult), we can amend the list of depths to $z_0, z_1, z_2, \ldots, z_n$ with $z_0 = 0$, and take v_k to correspond to the interval $[z_{k-1}, z_k]$. Then the vertical traveltime across this interval is $\Delta t_k = (z_k - z_{k-1})/v_k$ and the traveltime from z_0 to depth z_n is

$$t_k = \sum_{j=1}^{n} \frac{z_j - z_{j-1}}{v_j}, \tag{4.96}$$

where we would double this if we wanted the two-way time. Then, let $v_k(f)$ denote the frequency-dependent velocity for the kth layer as would be computed from Eq. (4.94), and the drift time is defined by

$$t_{\text{drift}} = 2\left[\sum_{j=1}^{n} \frac{z_j - z_{j-1}}{v_j(f_1)} - \sum_{j=1}^{n} \frac{z_j - z_{j-1}}{v_j(f_0)}\right], \tag{4.97}$$

where we take f_0 to be the well-logging frequency and f_1 to be the dominant frequency of some seismic dataset. The function `vint2t` performs the traveltime computation of

Code Snippet 4.7.9 This is an illustration of the computations of drift time required to create Figure 4.15. Line 1 loads a matfile containing well-log data from a well in western Canada. In this file are vectors `vp` and `z`, which give a list of P-wave velocities and depths as logged. Lines 2–6 prepare for the computation loop, which loops three times over the prescribed Q values. Line 9 performs the frequency adjustment of the velocities, moving them from frequency `f0` to frequency `f1`. Lines 11 and 13 compute the two-way vertical traveltime at these frequencies by calling *vint2t*. On line 15, the drift time is computed as the difference of these two times.

```
1   load data\logdata %contains vectors vp and z
2   Q=[50 80 120];
3   tdr=zeros(length(z),length(Q));
4   f1=30;%%seismic frequency
5   f0=12500;%logging frequency
6   z=z-z(1);%set first depth to zero
7   for k=1:length(Q)
8       %adjust velocity to frequency f1
9       v1=vp./((1-(1./(pi*Q(k))).*log(f1/f0)));
10      %compute times at f0
11      t0=vint2t(vp,z);
12      %compute times at f1
13      t1=vint2t(v1,z);
14      tdr(:,k)=t1-t0;%will be a positive quantity if f0>f1
15  end
```

End Code

wavepropcode/compute_tdrift.m

Eq. (4.96) for each depth in the list. Thus, there is a traveltime computed from $z = 0$ to every depth.

Figure 4.16a shows the evolution of an initial impulse in a 1D medium under constant-Q theory. The wavelets are computed from Eq. (4.92) followed by an inverse Fourier transform. This is accomplished by the function *einar* (Einar is Kjartansson's first name) using a velocity of 2000 m/s referenced to a frequency of 12 500 Hz. The wavelets were computed at a time sample interval of 0.0005 s and hence have no frequencies higher than 1000 Hz. Preceding each wavelet is a + sign marking the expected traveltime at 2000 m/s. The delay of these wavelets relative to the + sign is the drift. Figure 4.16b is similar to Figure 4.16a except that most of the delay has been removed and the wavelets have been normalized in amplitude. This permits an enlarged view for easier comparison. It is clear that the further the wavelet propagates, the more extended is the "blob" shape, which implies that the frequency content is lower.

The wavelets of Figure 4.16a do not look much like the wavelets that we have previously encountered. This is because the initial wavelet is a perfect impulse containing all frequencies, including 0 Hz. Since the mean value of a wavelet in the time domain equals its spectrum at 0 Hz, these wavelets are all positive and do not oscillate. More realistic wavelets can be easily computed by conducting a simulation with an initial wavelet that

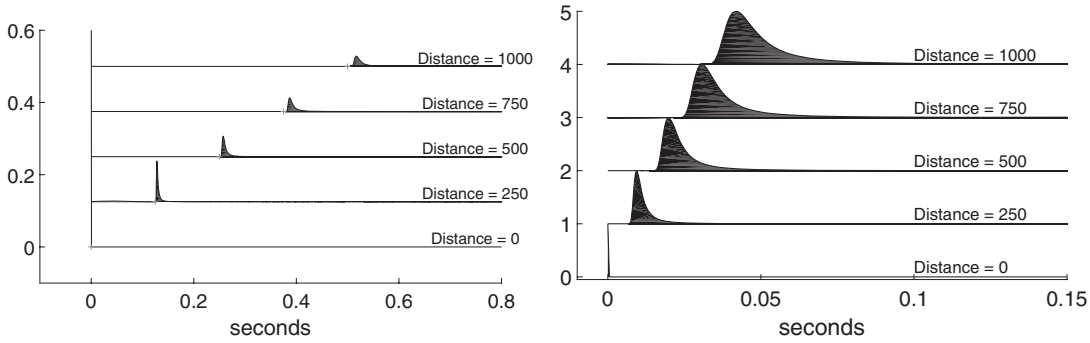


Figure 4.16a (left) Starting with an initial impulse, wavelets corresponding to the propagation distances [250, 500, 750, 1000] m are shown as predicted by constant-*Q* theory (Kjartansson, 1979). The velocity was 2000 m/s at 12 500 Hz and the discrete wavelets were sampled at 0.0005 s. The + sign preceding each wavelet indicates the predicted position at 2000 m/s.

Figure 4.16b (right) Similar to Figure 4.16a except that the wavelets have all been normalized to unit amplitude and most of the delay has been removed. This allows a more detailed view.

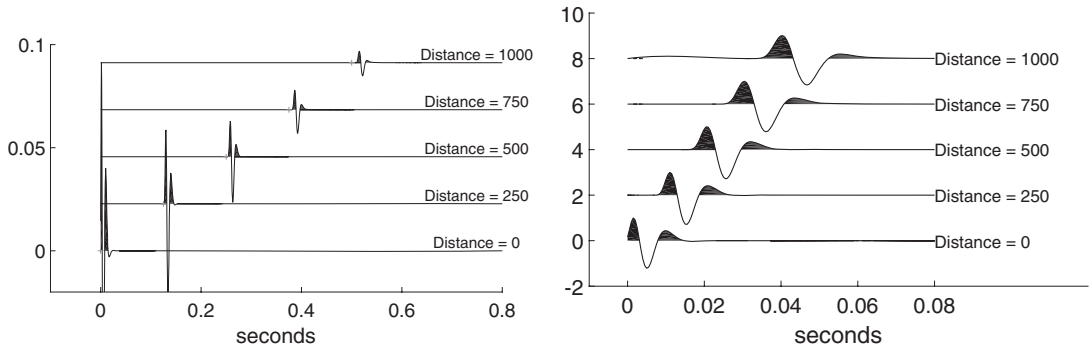


Figure 4.17a (left) Starting with an initial minimum-phase wavelet, wavelets corresponding to the propagation distances [250, 500, 750, 1000] m are shown as predicted by constant-*Q* theory (Kjartansson, 1979). The initial wavelet had a dominant frequency of 100 Hz. Compare with Figure 4.16a.

Figure 4.17b (right) Similar to Figure 4.17a except that the wavelets have all been normalized to unit amplitude and most of the delay has been removed. This allows a more detailed view. Compare with Figure 4.16b.

does oscillate, as might be emitted by a real seismic source. This is easily done by simply constructing a minimum-phase wavelet with some chosen dominant frequency and then convolving each of the wavelets of Figure 4.16a with this new wavelet. Figure 4.17a shows the result when the initial wavelet had a dominant frequency of 100 Hz, while Figure 4.17b shows the comparable delay-removed and normalized view. It is no accident that these wavelets all appear to be minimum phase. The initial wavelet was minimum phase, and it can be shown that the *Q* process is also. Futterman (1962) showed that any linear, causal attenuation process must always be minimum phase, and this result applies

to Kjartansson's theory. This is both good news and bad news. It is good because deconvolution theory relies upon minimum phase to estimate the wavelet, but the bad news is that any seismic trace will contain a complex superposition of many different minimum-phase wavelets corresponding to the differing traveltimes to each reflector. Close examination of the wavelets in Figure 4.17b shows that as propagation distance increases, the frequency content reduces and the phase also rotates subtly. While the phase change is less obvious than the frequency reduction, theory says the phase must change because amplitude and phase are linked by the Hilbert transform relation. If the amplitude spectrum changes, then the phase spectrum must change also.

Figure 4.18 shows the amplitude spectra of the wavelets of Figure 4.17a and the progressive decay of the dominant frequency is apparent. These amplitude spectra are described in theory as the product of the spectrum of the initial wavelet and the spectrum of the Q impulse response of Eq. (4.95), that is,

$$\left|\hat{w}\right|(f,t) = |\widehat{w_0}|(f)e^{-\pi ft/Q}, \tag{4.98}$$

where $|\widehat{w_0}|$ is the initial amplitude spectrum. Thus, as a wavelet propagates in a constant-Q medium, the initial amplitude spectrum is being constantly modified according to Eq. (4.98) and, because the process is minimum phase, the phase spectrum is also continuously evolving. In Figure 4.18, the spectrum at any distance is formed from the spectrum at distance 0 times $e^{-\alpha f}$, where $\alpha = \pi t/Q = \pi x/(qv_0)$. Thus there is exponential decay of the spectrum, and this is what moves the dominant frequency continually lower. Since the decay operator equals 1 at $f = 1$, all of the spectra intersect there. Similarly, if we focus

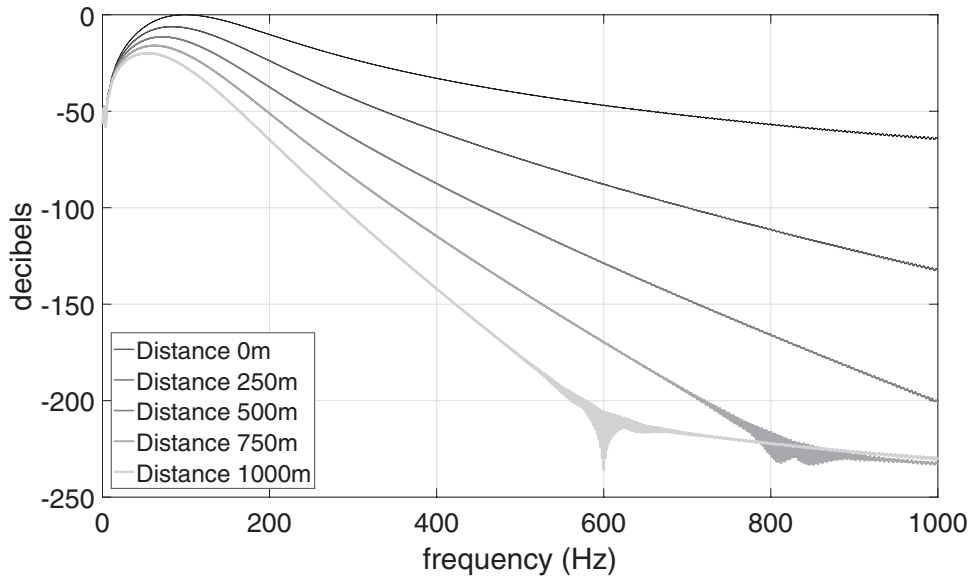


Figure 4.18 The amplitude spectra of the wavelets of Figure 4.17a. The initial wavelet had a dominant frequency of 100 Hz, which is clearly visible here. The dominant frequency progressively lowers as the propagation distance increases.

Table 4.1 Maximum signal frequency in Figure 4.18

Distance (m)	Time (s)	Max. signal frequency (Hz)
0	0	650
250	0.125	325
500	0.250	250
750	0.375	200
1000	0.5	170

attention on a single frequency, then that frequency decays exponentially with time according to $e^{-\beta t}$, with $\beta = \pi f/Q$. In any real case, we cannot expect to observe this exponential decay over more than a limited frequency or time range, because there will always be some background noise level. In Figure 4.18, there is a tremendous dynamic range of over 200 dB, whereas modern seismic systems rarely have a range much greater than 100 dB. Moreover, the background noise level is often around -50 to -80 dB. Suppose the noise level is -50 dB; then in Figure 4.18, the signal band must be a decreasing function of time according to Table 4.1, where the maximum signal frequency has simply been read from the figure as the frequency where each curve hits -50 dB. Other choices of noise level lead to different results, but the conclusion is obvious. The presence of attenuation means not only that the seismic wavelet is evolving toward lower frequency but also that the signal band, where signal stands above noise, must be decreasing. The term *nonstationary* is used to refer to this circumstance. A nonstationary signal is one whose spectrum is changing strongly with time.

4.7.4 Convolutional Seismograms with Attenuation

In Section 3.3.1, the concept of a convolution matrix was introduced as both a convenient way to visualize the convolution process and as a numerical device to accomplish the construction. Equation (3.53) shows that viewing it as matrix multiplication allows an understanding of the process. Consider the convolution $s = (w \bullet r)(t)$ and assume for convenience that w is of length 3, $w = [w_0, w_1, w_2]$, r is of length N, and $r = [r_0, r_1, r_2, \ldots, r_{N-1}]$, so that s will be of length $N + 2$. The convolution can be viewed as the matrix multiplication

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{N+2} \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ 0 \end{bmatrix} r_0 + \begin{bmatrix} 0 \\ w_0 \\ w_1 \\ \vdots \\ 0 \end{bmatrix} r_1 + \begin{bmatrix} 0 \\ 0 \\ w_0 \\ \vdots \\ 0 \end{bmatrix} r_2 + \ldots \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ w_3 \end{bmatrix} r_{N-1}. \tag{4.99}$$

Here, each column vector contains the wavelet with its $t = 0$ sample aligned with the corresponding reflectivity sample that scales the column. Assembling the columns into a matrix that multiplies the column-vector reflectivity gives the matrix view of convolution,

as shown in Figure 4.19 and discussed in Section 3.3.1. This view of seismogram construction can be easily extended to the nonstationary constant-Q case by simply placing the appropriate evolving wavelet in each column of the matrix. The resulting matrix is called a Q matrix to distinguish it from the stationary convolution matrix. This is shown in Figure 4.20 and should be compared closely with Figure 4.19. The evolving wavelet starts out as the same wavelet as used in the stationary case but then rapidly evolves into something else as prescribed by the constant-Q theory. The function `qmatrix` builds the Q matrix and requires as input a Q value and a starting wavelet. Both the convolution matrix and the Q matrix have the propagating wavelet in each column delayed to align with the appropriate reflection coefficient. However, in the convolution matrix the wavelet propagates without change, and this leads to the symmetry that the matrix is constant along any diagonal. In contrast, the Q matrix lacks this symmetry because the wavelet is continuously modified according to constant-Q theory. The matrix–vector multiplication then creates a nonstationary seismogram as a superposition of all of the columns of the Q matrix, each scaled by the appropriate reflection coefficient. Like the stationary convolutional model, the Q matrix model is a primaries-only construction neglecting the effects of multiples or transmission losses.

The Q matrix introduces both time-variant attenuation and drift into a seismogram. The former comes from the amplitude spectrum of the wavelet, while the latter comes from the phase, which is minimum. The wavelets forming the Q matrix are formed as the convolution between an initial wavelet and an `einar` wavelet. By default, the drift will be determined relative to the Nyquist frequency of the seismogram instead of

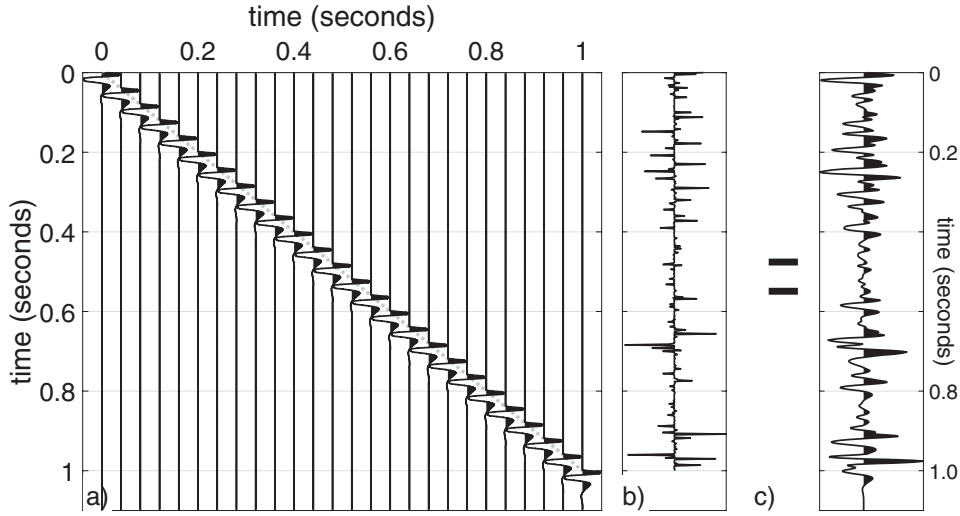


Figure 4.19 An illustration of the construction of a stationary seismogram by matrix multiplication. (a) The stationary convolution matrix with the same minimum-phase wavelet in each column, delayed to place its first sample on the diagonal (dashed line). Every twentieth column is shown. (b) The reflectivity signal, here 1 s long. (c) The stationary synthetic seismogram resulting from the convolution.

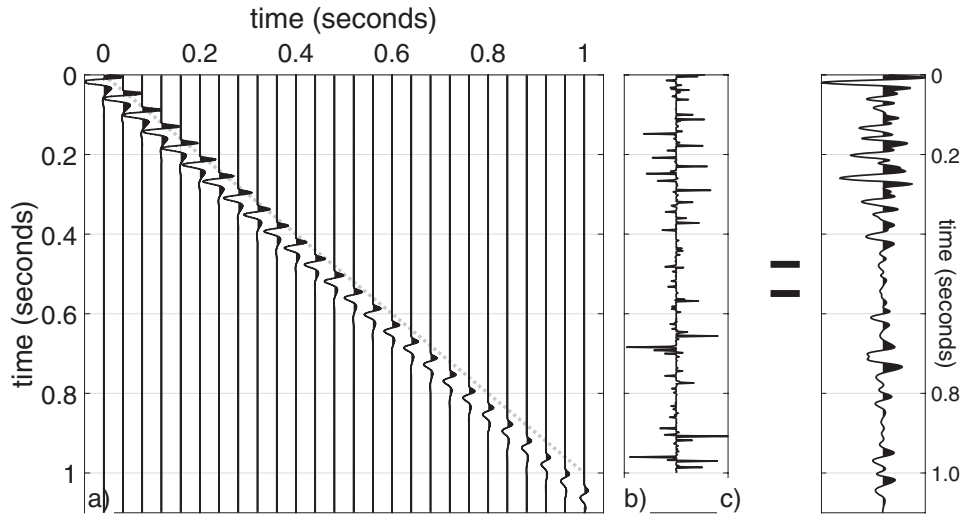


Figure 4.20 An illustration of the construction of a nonstationary seismogram by matrix multiplication. (a) The nonstationary Q matrix with an evolving minimum-phase wavelet in each column, delayed to place its first sample on the diagonal (dashed line) but showing that the main energy is progressively delayed more. Every twentieth column is shown. (b) The reflectivity signal, here 1 s long. (c) The nonstationary synthetic seismogram resulting from the convolution.

perhaps 12 500 Hz, as might be appropriate for well-derived velocities. This results in considerably less drift than might be the case with real data. The parameter `flag` (see Code Snippet 4.7.10) allows some flexibility with this. The option exists to compute the Q matrix with 12 500 Hz as the Nyquist frequency and then resample it (`flag=2`), or to simply estimate the additional drift and apply it as a time shift (`flag=3`). For more accurate control over this effect, one should consider using the *`vspmodelq`* method described in the next section. It is worth noting that velocities measured in well logging are often corrected by an empirical method called a *check shot*. This involves lowering a receiver into the well and directly recording a time–depth relationship at seismic frequencies. This information can then be used to drift-correct the logging measurements. If this has been done, then the drift should be greatly reduced.

Figure 4.21a compares a stationary and a nonstationary seismogram constructed from the same reflectivity. The code that makes these seismograms is shown in Code Snippet 4.7.10, where it can be seen that the same minimum-phase 30 Hz wavelet is used in the construction of both the convolution matrix and the Q matrix. The Q matrix also used a Q value of 50. The stationary and nonstationary seismograms are then constructed in similar fashion by matrix–vector multiplication with the reflectivity. The actual plot in Figure 4.21a was made with the utility *`trplot`*, and this code is not shown. (*`trplot`* is very useful for making displays and comparisons of single traces, but it is left to the reader to explore.) The initial parts of both seismograms are very similar because there has been very little attenuation, but as time increases the nonstationary seismogram shows a strong amplitude decrease. This is entirely due to the attenuation effect and not some

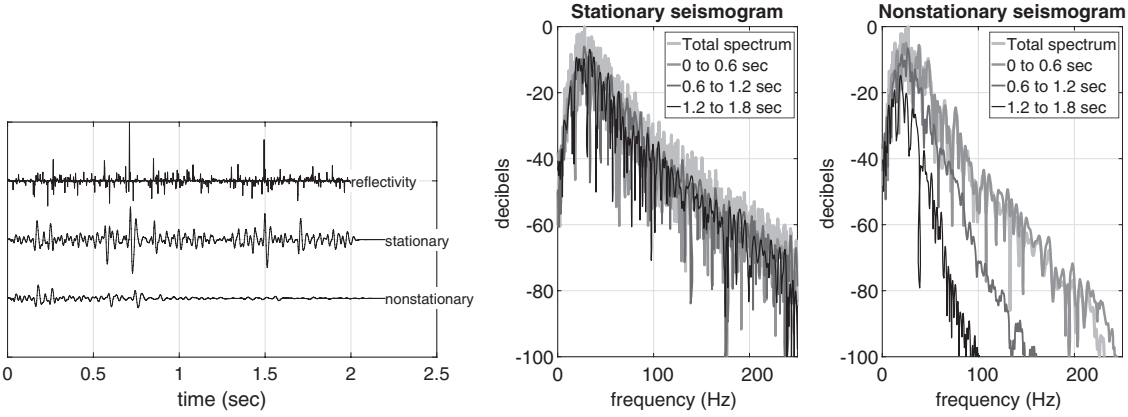


Figure 4.21a (left) A stationary and a nonstationary synthetic seismogram made from the same reflectivity. The stationary seismogram used a 30 Hz minimum-phase wavelet, while the nonstationary seismogram used $Q = 50$ and the same wavelet. See Code Snippet 4.7.10.

Figure 4.21b (right) Time-variant spectra for the synthetic seismograms of Figure 4.21a. In each case the total amplitude spectrum and the spectra in three 600 ms windows are shown. See Code Snippet 4.7.10.

Code Snippet 4.7.10 This illustrates the construction of both stationary and nonstationary seismograms from the same reflectivity. Line 7 constructs a convolution matrix from a minimum-phase wavelet, while line 8 uses the same wavelet and $Q = 50$ to construct a Q matrix. The seismograms are then constructed by matrix–vector multiplication with the reflectivity on lines 9 and 10. The results are displayed in Figure 4.21a in the time domain and Figure 4.21b in the frequency domain.

```
1   dt=.002;%time sample size
2   tmax=2;%maximum time
3   fdom=30;%dominant frequency of the wavelet
4   Q=50;%Q value
5   [r,t]=reflec(tmax,dt,.1,3,pi);%synthetic reflectivity
6   [w,tw]=wavemin(dt,fdom,.2);%min phase wavelet
7   cmat=convmtx(w,length(r));%a convolution matrix
8   qmat=qmatrix(Q,t,w,tw,3,2);%a Q matrix
9   s=cmat*r;%stationary synthetic
10  sn=qmat*r;%nonstationary synthetic
```

End Code

wavepropcode/ nonstat_synthetic .m

other process (such as wavefront spreading). The amplitude decay is the most obvious manifestation of the high-frequency attenuation inherent in the Q operator as shown in Eq. (4.98). Figure 4.21b gives more perspective by examining the amplitude spectra of the two signals. For each signal, the total amplitude spectrum (from 0 to 2 s) is shown together with the spectra in three 0.6 s windows: shallow, intermediate, and deep. (These

displays were made with the utility `tvdbspec`, which is left to the reader to investigate.) In the stationary case, the overall shapes of the four spectra are essentially similar. The detailed oscillations are caused by the reflectivity, while the overall shape comes from the wavelet. Since the wavelet is unchanging, the spectra are all similar. The spectra of the three windows are slightly below the total spectrum simply because the windows have less signal power than the entire signal. In the nonstationary case, the spectra in the individual windows are quite distinct from one another. The deeper windows show clear evidence of greater decay than the shallower ones. The total spectrum, rather than being some sort of average of the three windows, is more similar to the shallow spectrum because that has greater power than the two deeper windows. Figure 4.22 shows a different perspective on the three windowed spectra of the nonstationary signal in Figure 4.21b. The spectrum from each window is plotted on separate axes together with three spectra taken directly from the Q matrix. These three additional spectra are taken directly from the three columns of the Q matrix that correspond to the centers of the three analysis windows. Thus these are the spectra of the propagating wavelet at the center of each window. Figure 4.22 shows that the three windowed spectra, if smoothed, would give reasonable estimates of the spectra of the propagating wavelet at the window centers.

The Q matrix approach allows the first-order effects of constant-Q attenuation to be easily simulated. In the discussion thus far, the Q matrix was constructed using a single scalar Q value; however, a time-variant Q can also be accommodated by inputting a time-variant

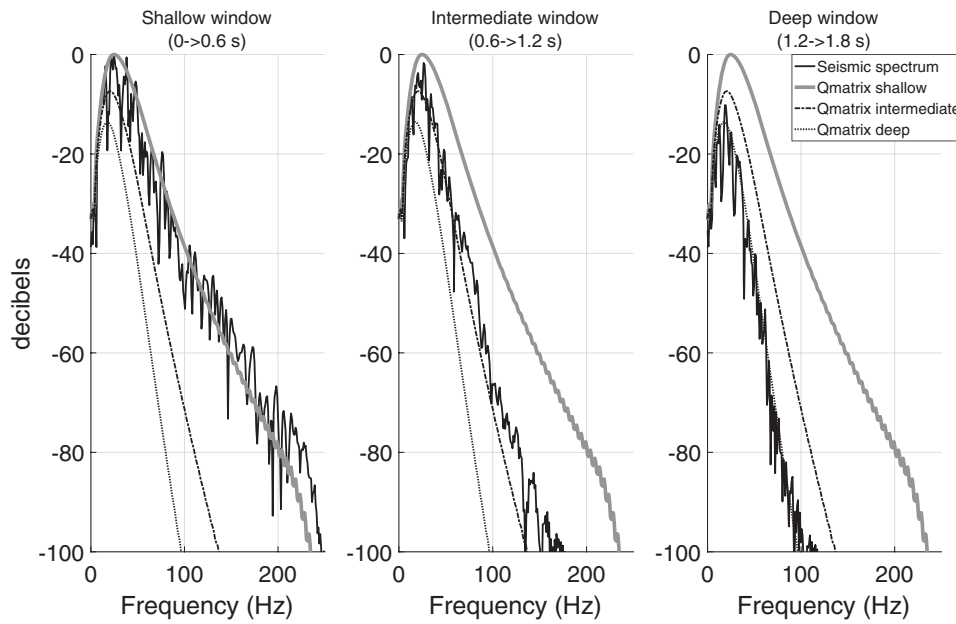


Figure 4.22 The three windowed spectra of the nonstationary seismogram of Figure 4.21b are shown on separate axes and compared with spectra from the Q matrix. The three Q matrix spectra are the spectra of the column of the Q matrix corresponding to the window center for the three windows.

average Q in the `qmatrix` function. This is appropriate for a 1D primaries-only simulation, where time and depth have a unique correspondence, but becomes more approximate in 2D or 3D. Consider a 1D layered Earth where the layers are characterized by vectors of velocity, time, and Q of length N, $\underline{v} = [v_0, v_1, \ldots, v_{N-1}]$, $\underline{t} = [t_0, t_1, \ldots, t_{N-1}]$, and $\underline{v} = [v_0, v_1, \ldots, v_{N-1}]$. Using Eq. (4.98), we can relate the amplitude spectrum at time t_1 to that at time t_0 by $\left|\hat{w}\right|(f, t_1) = \left|\hat{w}\right|(f, t_0)e^{-\pi f(t_1-t_0)/Q_0}$. Similarly, the amplitude spectrum at time t_2 is $\left|\hat{w}\right|(f, t_2) = \left|\hat{w}\right|(f, t_1)e^{-\pi f(t_2-t_1)/Q_1}$. We can combine these expressions by substituting the first into the second and combining the exponentials to get $\left|\hat{w}\right|(f, t_2) = \left|\hat{w}\right|(f, t_0)e^{-\pi f[(t_2-t_1)/Q_1+(t_1-t_0)/Q_0]}$. This process can be continued through an arbitrary number of layers to give an expression relating the spectrum at level k to that at level 0 as

$$\left|\hat{w}\right|(f, t_k) = \left|\hat{w}\right|(f, t_0)e^{-\pi f\sum_{j=0}^{k}(t_{j+1}-t_j)/Q_j}. \tag{4.100}$$

We define the average Q for the interval t_0 to t_k to be that required in the expression $\left|\hat{w}\right|(f, t_0)e^{-\pi f(t_k-t_0)/Q_{\text{ave}}}$ to make it equal to Eq. (4.100). Clearly this requires $(t_k - t_0)/Q_{\text{ave}} = \sum_{j=0}^{k}(t_{j+1} - t_j)/Q_j$, from which we deduce

$$Q_{\text{ave}}^{-1}(t_k, t_0) = \frac{1}{t_k - t_0}\sum_{j=0}^{k}(t_{j+1} - t_j)/Q_j. \tag{4.101}$$

The various Q_j are called *interval Qs* and we see that the interval Qs combine inversely, weighted by interval traveltime, to give the inverse average Q. If we happen to have average Q values to two different levels t_n and t_m with $t_n > t_m$, then we define the interval Q between these levels as

$$Q_{\text{int}}^{-1}(t_n, t_m) = \frac{1}{t_n - t_m}\left[\frac{t_n - t_0}{Q_{\text{ave}}(t_n, t_0)} - \frac{t_m - t_0}{Q_{\text{ave}}(t_m, t_0)}\right]. \tag{4.102}$$

Equations (4.101) and (4.102) make it possible to move between an interval Q description and an average Q description. The interval Q is really the average Q for the interval unless the interval happens to correspond to a homogeneous layer, in which case it is an intrinsic local Q. If we suppose that there exists such an intrinsic local Q, say $Q(z)$, with variations on the same scale (roughly 1 ft sampling) as those seen in sonic and density logs, then the local interval Q for the interval $[z_m, z_n]$ is computed from $Q(z)$ by first using velocity information to convert $Q(z)$ to $Q(t)$ and $[z_m, z_n]$ to $[t_m, t_n]$ and then computing

$$Q_{\text{int}}^{-1}(t_n, t_m) = \frac{1}{t_m - t_n}\sum_{t_k=t_m}^{t_n}\frac{t_{k+1} - t_k}{Q_k}, \tag{4.103}$$

where $Q_k = Q(t = t_k)$. The functions `qint2qave`, `qave2qint`, and `qz2qint` are found in the *NMES Toolbox* and provide the functionality of Eqs. (4.101), (4.102), and (4.103). Also found is the function `fakeq`, which allows the creation of a plausible $Q(z)$ from sonic and density logs via a proposed empirical relationship. The interested reader should type `help qtools` to see more tools of this type, and further investigation is left to the reader.

Adding random noise to a nonstationary seismogram is easily done using *rnoise*, as described near the end of Section 4.7.2. The one additional consideration is that the magnitude of the noise, as determined by the specified signal-to-noise ratio, should be specified in some particular time zone of interest. A nonstationary seismogram with noise will then have a time-variant signal-to-noise ratio which will have the specified value in the zone of interest and a greater value at earlier times and a lesser value at greater times.

A final remark on the Q matrix approach is that it can be used to modify any synthetic seismic data that has been created without attenuation to include the first-order effects of attenuation. For example, a shot record modeled with the finite-difference tools discussed in this chapter can be modified to include attenuation by creating a Q matrix and applying it to each trace in the shot record. In this context, it is important to generate the Q matrix with an impulse wavelet (just $w = [1, 0]$ and $tw = [0, \delta t]$ will do) so that the wavelet is not applied twice. Also, using a single scalar Q value is most correct in this context.

Exercises

4.7.3 Repeat the computation in Code Snippet 4.7.10 for all three possible values of the parameter `flag` in *qmatrix*. Compare the results with each other and with the stationary seismogram. What is the maximum drift that you observe in each case?

4.7.5 Seismograms with Internal Multiples and Attenuation

There are two functions in the *NMES Toolbox* that facilitate the creation of high-fidelity 1D seismograms with multiples, and these are *seismo* and *vspmodelq*. The former is found in the *syntraces* folder, while the latter is in *qtools*. The function *seismo* creates a seismogram using the Goupillaud method described in Section 4.6.3. This is a time-domain method that works directly from sonic and density logs to create a 1D, lossless P-wave seismogram. Outputs include the reflectivity in time, the seismogram with multiples, and the separated primaries and multiples. Attenuation is not included, but can be applied afterwards using the Q matrix approach discussed in Section 4.7.4. The function *vspmodelq* is a frequency-domain approach that does not block the logs into equal-traveltime layers, but instead uses them exactly as input. It requires velocity, density, and intrinsic local Q logs and produces a complete 1D vertical seismic profile (VSP), assuming a source at $z = 0$. A result comparable to that of *seismo* comes from a receiver specified at $z = 0$ in *vspmodelq*. Complete control of multiples and transmission losses is provided. A similar function, *vspmodelqs*, provides the ability to place the source at depth.

Code Snippet 4.7.11 shows an example of running *seismo* using the well-log data provided with this book (see Figure 4.15 for a plot of the P-wave velocity log). The required logs are sonic and density logs and they must be of the same length. Either a displacement or a pressure seismogram can be produced (see Section 4.6.2 for a discussion of the difference), and multiples can be turned on or off. A primaries-only seismogram has a definite, finite length in time, and that is determined by the two-way traveltime to the bottom of the

Code Snippet 4.7.11 This is an example of running *seismo* with well-log information to create a 1D seismogram with multiples. Line 2 loads the log data from a prepared .mat file. Lines 3–7 define parameters for *seismo*. Line 8 adjusts the depths to start from 0 because this causes the calculated times to also start from 0. Line 9 sets the maximum time to be slightly larger than that corresponding to the end of the logs to show better the multiples. Line 10 runs *seismo*, and the temporal sample rate is determined from that of the wavelet built on line 5. The first output from *seismo*, spm, is the seismogram including both primaries and multiples, and t is its time coordinate. The other outputs, rcs, pm, p, are the reflectivity, primaries + multiples, and primaries only with transmission losses. All outputs are time-domain and of identical length. Lines 11 and 12 create two additional seismograms by simple convolution.

```
1  %file logdata contains sp (p-sonic), rho (density) and z (depth)
2  load data\logdata
3  dt=.001; %sample rate (seconds) of wavelet and seismogram
4  fdom=60;%dominant frequency of wavelet
5  [w,tw]=ricker(dt,fdom,.2);%ricker wavelet
6  fmult=1;%flag for multiples. 1 multiples, 0 no multiples
7  fpress=0;%flag, 1 pressure (hydrophone), 0 displacement (geophone)
8  z=z-z(1);%adjust first depth to zero
9  tmin=0;tmax=1.0;%start and end times of seismogram
10 [spm,t,rcs,pm,p]=seismo(sp,rho,z,fmult,fpress,w,tw,tmin,tmax);
11 sp=convz(rcs,w);%make a primaries only seismogram
12 sp2=convz(p,w);%primaries with transmission losses
```

End Code

wavepropcode/seismo_example.m

logs. In contrast, when multiples are turned on, the temporal length is effectively infinite because there is no limit to the number of bounces (reflections) that can occur. However, since reflection coefficients are small numbers (usually less than 0.1 in absolute value) and since the amplitude of an n-bounce multiple is the product of n reflection coefficients and a number of transmission coefficients, the amplitude of multiples decreases rapidly with n. As a time-domain algorithm, *seismo* cannot compute infinite traveltime, so the length of the multiple train that is realized is controlled by the tmax parameter on line 8. By default, tmax is set to the two-way traveltime to the deepest reflector, which in this case is about 0.8 s. Here it is set to 1.0 to show a bit of the multiple train. The result of this computation is shown in Figure 4.23a in the time domain, and the spectra are shown in Figure 4.23b. In the top part of Figure 4.23a are the reflectivity series and three seismogram products, all displayed with a "spike wavelet" or essentially no wavelet at all. The reflectivity series is that computed by the Goupillaud method of defining layers with constant traveltime thickness, which is 0.001 s in this case. The trace labeled "primaries and multiples" is the essential output of the Goupillaud algorithm as illustrated in Figure 4.6. The function *seismo* also outputs the trace labeled "primaries with transmission loss," which contains only the primary reflections but each is reduced in amplitude by the accumulated two-way transmission loss for the overlying interfaces (see Eq. (4.76)). Close comparison

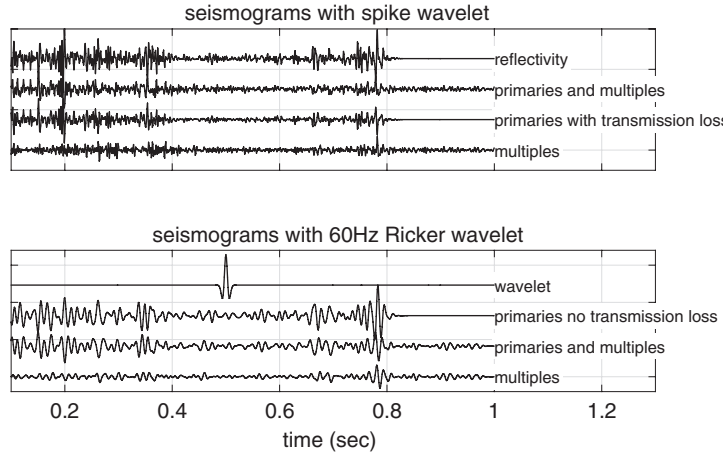


Figure 4.23a Synthetic seismograms as created by the *seismo* method as shown in Code Snippet 4.7.11. In the top panel are shown the results when the wavelet is a perfect impulse, while in the bottom panel are the results for a 60 Hz Ricker wavelet.

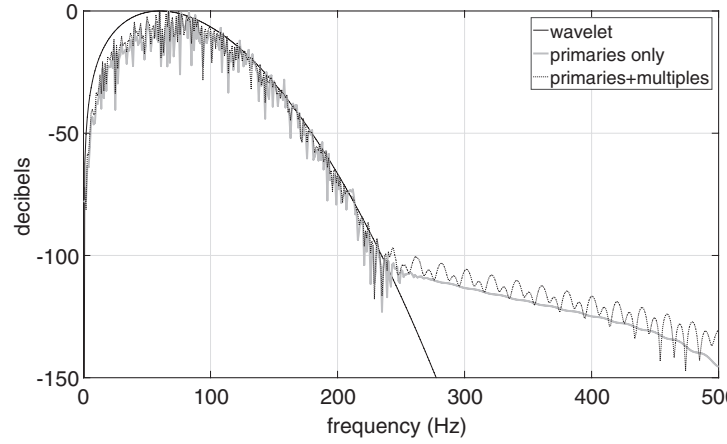


Figure 4.23b Spectra of the wavelet and two of the seismograms from the bottom panel of Figure 4.23a.

of the primaries-with-transmission-loss response and the reflectivity shows that the former differs in sign and amplitude from the latter for each reflection. The sign difference is because this is a displacement seismogram, where the reflected wave u_{R} is always related to the incident wave u_{I} by $u_{\mathrm{R}} = -Ru_{\mathrm{I}}$ (where R is the reflection coefficient). For a pressure seismogram, the reflection law does not have the minus sign (see Section 4.6.2). The final trace in this figure, labeled "multiples," is computed as the subtraction of primaries-with-transmission-loss from primaries-and-multiples. Notice that the multiples extend to far greater times than the primaries and, in theory, go on forever. Examining the spectra in Figure 4.23b shows an enormous dynamic range of more than 100 dB and a very high-frequency response. This dynamic range is a key feature of this type of algorithm and this is why it is considered a high-fidelity method. It is very difficult to get anything approaching

this dynamic range from something like the finite-difference method. Also of interest is the observation that the spectral peak of the seismograms is about 75 Hz, while the dominant frequency of the Ricker wavelet was only 60 Hz. (See Exercise 4.7.4.)

The reflectivity series of Figure 4.23a is 829 samples long (not counting the zeros at the end), while the well log is 10 890 samples. This is about a 13 to 1 downsampling and is a consequence of the Goupillaud equal-traveltime blocking at 0.001 s. Figure 4.24 compares the Goupillaud reflectivity with that computed directly from the well logs in depth and then displayed in time. It is apparent that there are both similarities and differences. It may be concerning to see such a drastic downsampling, but experience shows that the consequences are nearly negligible within a typical seismic bandwidth. This is illustrated in Figure 4.25, which compares seismograms computed for a sequence of time sample sizes ranging from $\Delta t = 0.004$ to $\Delta t = 0.0005$ s. In `seismo`, Δt also controls the Goupillaud traveltime blocking and it is apparent that there is very minimal effect. Close inspection shows that there is a slight increase in transmission loss as Δt decreases, but this is balanced by a slight increase in multiple content.

The function `vspmodelq` offers greater flexibility and more sophistication than `seismo`. It is based on Ganley (1981) and discussed in detail in Margrave and Daley (2014). Contained in the *NMES Toolbox* is the script `test_vspmodelq`, which illustrates the use of `vspmodelq` in great detail and will be discussed here only briefly. Compared with `seismo`, `vspmodelq` computes a complete 1D VSP, which models a common experiment where a source is placed near a borehole and recorded into receivers placed at specific depths in the borehole. The VSP trace for a receiver at $z = 0$ is a synthetic seismogram directly comparable with that obtained from `seismo`. Also, `vspmodelq` directly includes

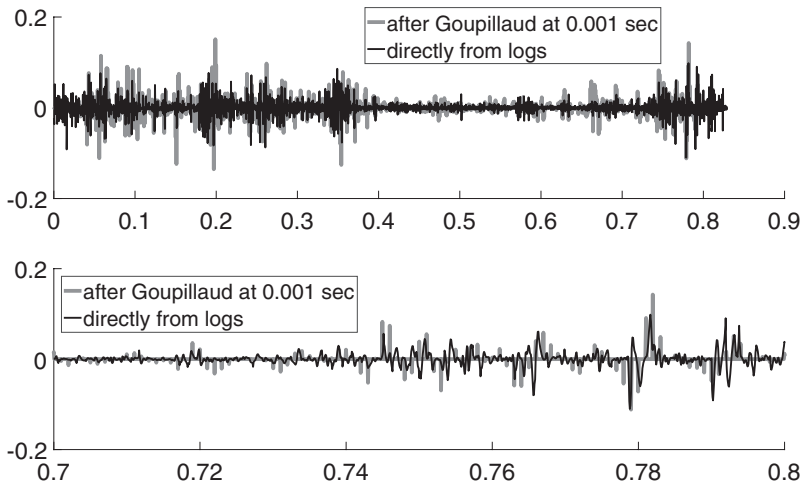


Figure 4.24 The top panel shows the reflectivity series estimated by the Goupillaud method with 0.001 s equal-traveltime layering compared with the actual well-log reflectivity, computed in depth and then displayed in time. There are 10 889 samples in the latter but only 829 samples in the former. The bottom panel is an enlargement of a portion of the top panel.

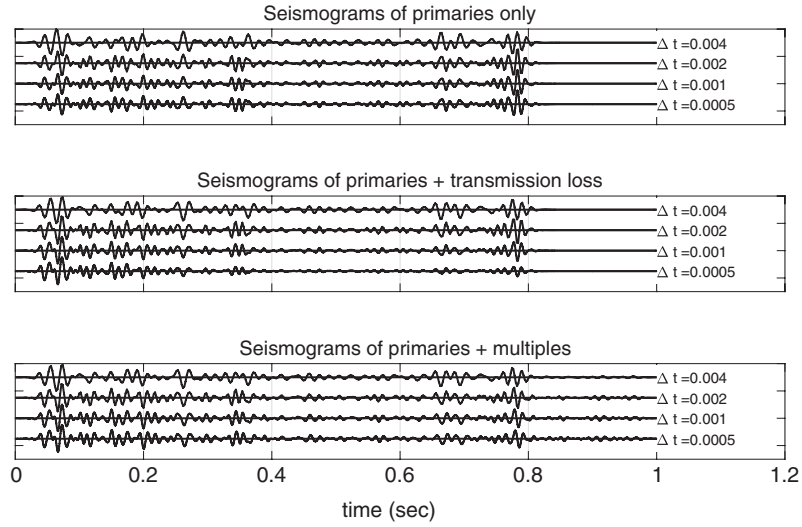


Figure 4.25 This is a comparison of results from `seismo` for a range of different Goupillaud time thicknesses indicated by Δt. All seismograms have the same 50 Hz Ricker wavelet. There is remarkably little effect on the final results as Δt changes by an order of magnitude. The seismograms have all been normalized to have the same maximum amplitude.

constant-Q theory and allows the input of either a single scalar Q value or a complete log of Q values. Unlike `seismo`, this is a frequency-domain method, and this has both positive and negative implications. On the positive side, frequency-dependent velocity and frequency-dependent reflectivity are both easily modeled. However, a negative consequence is that all possible multiples are always computed, including those with infinite traveltime. The digital computation of a finite-length seismogram means that there can be strong and objectionable time-domain wraparound of these multiples. The best way to suppress them is to include a strong Q effect, which will mostly suppress them. Also significant is that `vspmodelq` does not invoke Goupillaud equal-traveltime blocking but instead uses the model exactly as specified by the input logs. Useful companion functions are `blocklogs` and `fakeq`. The function `blocklogs` accepts an LAS (Log ASCII Standard) as input, and outputs velocity and density vectors, reblocked as desired, and with an overburden attached. The function `fakeq` accepts as input the velocity and density vectors specifying a model and creates an equal-length Q vector based on an assumed empirical relation between velocity, density, and Q. This relation assumes that both velocity and density influence the local Q through linear relations. To specify these relations, the user specifies two velocity–Q pairs and two density–Q pairs, and since two points define a linear relation, these define linear relations between velocity, density, and Q. The idea is that less dense rocks and lower-velocity rocks will be less structurally competent and hence have lower Q. So, for example, the velocity–Q pairs might be $(v_1, Q_1) = (1000, 20)$ and $(v_2, Q_2) = (4000, 200)$ and similarly for density. Then, if Q_v is the velocity determined from Q and Q_ρ is that determined from density, then the final Q is $Q^{-1} = Q_v^{-1} + Q_\rho^{-1}$. In

this way, the Q vector computed from `fakeq` is as detailed as the velocity and density vectors and is influenced by both. At this time, there is no known field technique to measure Q with the detail of a well log, so this approach is a plausible method to prescribe for such modeling.

Exercises

4.7.4 Examine Figure 4.23b closely and notice that the dominant frequency of either seismogram is slightly higher than that of the 60 Hz Ricker wavelet. Repeat the computation in Code Snippet 4.7.11 and recreate this figure. Use `domfreq` to determine numerical values for the three dominant frequencies. Then investigate further until you have a theory for why the seismograms have a higher dominant frequency than the wavelet. Explain your theory.

4.7.5 Repeat the computation of Code Snippet 4.7.11 for both displacement and pressure seismograms. Verify that the "primaries only" responses have the opposite polarity. Create convincing figures and explain why this is the case.

4.8 Chapter Summary

This chapter has presented a survey of wave propagation theory and seismic modeling methods that was not intended to be complete, but rather to give an overview and an understanding of basic concepts and the types of modeling methods in common use. The discussion began with a derivation of a scalar wave equation as it arises from physical considerations for a vibrating string or an inhomogeneous fluid. Of course, seismic waves are vector waves, but there is much to be learned from scalar waves and almost all seismic processing methods ignore the vector nature.

Following next was a presentation of the solution of the scalar wave equation by finite-difference time-stepping and the introduction of MATLAB tools for this purpose. The fundamental features of finite-difference solutions, such as stability and dispersion, were identified and consequent modeling parameter choices were suggested.

Next the 1D normal-incidence seismogram was introduced as a method capable of computing all possible multiples and one that is widely used in industry. Considerable attention was paid to identifying the contribution of primaries and multiples to the final seismogram and to the difference between displacement and pressure constructions. MATLAB tools for the computation were introduced and demonstrated.

Finally, this chapter discussed how to modify seismograms to include anelastic attenuation as described by constant-Q theory. The construction of the Q impulse response was discussed and the concept of the Q matrix was introduced. The Q matrix generalizes a wavelet convolutional matrix to the case of a wavelet that constantly evolves owing to an attenuation process. The application of a Q matrix to a reflectivity was shown to give a realistic nonstationary seismogram.

5 Deconvolution: The Estimation of Reflectivity

Like many seismic processing algorithms, *deconvolution* means different things to different people. To some, it is the name given to a category of techniques that are designed to convert seismic traces into sequences of *reflection coefficients*. This requires correction for a number of effects, including source and receiver signatures, anelastic attenuation, multiples, and possibly other effects. These reflection coefficients are then positioned properly in space by *migration*. However, to others, deconvolution means only an algorithm designed to remove the source signature. Between these extremes are many intermediate viewpoints. This spectrum of perspectives means that there is a great deal of confusion and misinformation surrounding the subject. In this book, deconvolution is viewed in the broadest sense as a process with the ultimate goal of the estimation of reflection coefficients. Since the removal of the source signature is necessary for this purpose, the narrow view can be considered as part, but only part, of the total. Also, we will address the need to run preprocessing steps on raw seismic traces before deconvolution.

All of the deconvolution processes discussed in this chapter must confront the same fundamental dilemma: the source signature and other effects to be removed are not known in detail and must be estimated from the data. These methods are sometimes called *blind deconvolution* in reference to the fact that the wavelet to be deconvolved is not known a priori. The effects to be removed are typically lumped together into a single convolutional operator, called the *wavelet*. These deconvolution techniques are also called *statistical* because their success depends upon assumed statistical properties of the data. The successful estimation of the wavelet requires a viable mathematical model of a seismic trace whose key components have the expected statistical properties. The *convolutional model*, discussed in the next section, is the most common trace model and encapsulates the assumptions behind most statistical deconvolution algorithms. The convolutional model has already been encountered in Section 2.3.1 and again in Section 4.7.2. Here the intent is to more clearly relate the model to the wave equation and to better understand its approximate nature.

5.1 The Deconvolution Trace Model

We will consider two alternative formulations which could both be called a convolutional model for the seismic trace. The first model is mathematically exact but not very useful, while the second model is inexact and somewhat intuitive but very useful.

Ultimately, seismic data comes from wave motion, so there should be a wave equation basis to a trace model. The first formulation is based on the idea of a Green's function, which is the solution to a wave equation whose parameters represent the Earth and where the seismic source term is an idealized impulse. The temporal recording (at a geophone, for instance) of this wavefield is called the Earth's impulse response. Then this first trace model says that the seismic response (i.e., the trace) resulting from a real-world source is the convolution of the temporal waveform emitted by the source, called the *wavelet*, with the aforementioned impulse response. In this text, this model will be called the *Green's function model* of the seismic trace and the term *convolutional model* will be reserved for the second trace model.

The second formulation postulates that the seismic trace can be modeled as the convolution of the source wavelet not with the Earth's impulse response but with another function, called the Earth's *reflectivity*. The reflectivity is a function of time, t, whose value is the Earth's reflection coefficient for a position whose traveltime from the receiver is t. This definition is nonunique, since the locus of points whose traveltime from the receiver is t is a hemisphere (assuming constant velocity) centered on the receiver. Moreover, if we assume travelpaths that are not direct but can bounce from one reflector to another, then this becomes even more confusing. Therefore, the reflectivity will be defined as a function of time whose value at t is the Earth's reflection coefficient for the point vertically beneath the receiver at two-way traveltime t. This definition is appropriate for most sedimentary basins and we can consider that the reflectivity is directly measurable by sonic and density logs in boreholes (although the measurement must be converted from depth to time).

While the two models are superficially similar, it is the second model that is widely used and which leads to relatively simple deconvolution algorithms. Since the first model has a strong theoretical justification, this suggests that we need to understand the relationship between impulse response and reflectivity, and perhaps this will lead to better strategies for creating reflectivity estimates.

5.1.1 The Green's Function Model

Many effects of a linear 1D Earth can be modeled with the scalar wave equation

$$\frac{\partial^2 u(z,t)}{\partial z^2} - \frac{1}{v_{\text{ins}}^2(z)} \frac{\partial^2 u(z,t)}{\partial t^2} = f(z,t), \tag{5.1}$$

where u is the seismic wavefield, $v_{\text{ins}}(z)$ is the instantaneous velocity profile, and $f(z,t)$ specifies the source function. The source function is essentially arbitrary and specifies the location and temporal signature of any sources. Equation (5.1) can be solved if the solution to a similar equation with an impulsive source is known. This required auxiliary function is called a *Green's function*, $g = g(z,\zeta,t,\tau)$, and is the solution to

$$\frac{\partial^2 g}{\partial z^2} - \frac{1}{v_{\text{ins}}^2(z)} \frac{\partial^2 g}{\partial t^2} = \delta(z-\zeta)\delta(t-\tau). \tag{5.2}$$

In this equation, the source is represented with Dirac delta functions as an idealized impulse at $z = \zeta$ and $t = \tau$. Since the equation is invariant in t, we can rewrite $g = g(z, \zeta, t - \tau)$ as a function of only three variables. Given $g(z, \zeta, t - \tau)$ and $f(z, t)$, the solution to Eq. (5.1) is found by the convolution

$$u(z,t) = \iint_{-\infty}^{\infty} g(z,\zeta,t-\tau) f(\zeta,\tau)\, d\zeta\, d\tau. \tag{5.3}$$

To see that Eq. (5.3) solves Eq. (5.1), note that the z and t dependence occurs only in g and that the integration is with respect to ζ and τ. Substituting Eq. (5.3) into Eq. (5.1) and moving the second derivatives with respect to z and t under the integral sign, Eq. (5.1) becomes

$$\begin{aligned} &\iint_{-\infty}^{\infty} \left[\frac{\partial^2 g(z,\zeta,t-\tau)}{\partial z^2} - \frac{1}{v_{\text{ins}}^2(z)} \frac{\partial^2 g(z,\zeta,t-\tau)}{\partial t^2} \right] f(\zeta,\tau)\, d\zeta\, d\tau \\ &\qquad = \iint_{-\infty}^{\infty} \delta(z-\zeta)\delta(t-\tau) f(\zeta,\tau)\, d\zeta\, d\tau = f(z,t). \end{aligned} \tag{5.4}$$

In this equation, the transition from the first form to the second uses Eq. (5.2) and the final form uses the sifting property of the Dirac delta function (Eq. (2.26)). Now let the source function, $f(z, t)$, be modeled as a point source at the spatial coordinate origin but with an extended response in time, $f(z, t) = \delta(z)w(t)$, where $w(t)$ is the *source waveform*, or *wavelet*. Furthermore, let a seismic trace, $s(t)$, be modeled as the surface recorded wavefield $u(z = 0, t)$. Then Eq. (5.3) reduces to the *Green's function model* for a seismic trace,

$$\begin{aligned} s(t) = u(z=0,t) &= \int_{-\infty}^{\infty} g(z=\zeta=0, t-\tau) w(\tau)\, d\tau = (g(z=\zeta=0,\cdot) \bullet w(\cdot))\,(t) \\ &= g(z=\zeta=0,t) \bullet w(t), \end{aligned} \tag{5.5}$$

where the third and fourth forms are equivalent, but the third form is mathematically more proper as it states that the t dependence occurs in the final convolved function; the simpler fourth form is commonly found in geophysics writing.

It is tempting to consider this result (Eq. (5.5)) to be the basis of the widely used convolutional model; but, in reality, it is not a very useful starting point for a deconvolution algorithm. The equation states that the wavefield that results from a spatial point source emitting a temporal waveform $w(t)$ is obtained by convolution of $w(t)$ with the Green's function $g(z = \zeta = 0, t)$. The problem with this result is that $g(z, \zeta, t)$ is a very complex function in general. Intuitively, $g(z, \zeta, t)$ is the response of the real Earth to an impulsive source, and so it is called the *Earth impulse response*. Thus it contains all physical effects such as reverberations, transmission losses, surface waves, and more. If $w(t)$ were successfully deconvolved from $s(t)$, the result would still be very difficult to interpret. Even more problematic, when $w(t)$ is unknown (the most common case) and must be estimated from $s(t)$ itself, then the simple statistical arguments that we will employ in the following sections are not overtly valid.

5.1.2 The Convolutional Model

A simpler and more useful trace model than the Green's function model is the *convolutional model*. In its simplest form, this model postulates that a seismic trace can be represented as

$$s(t) = \int_{-\infty}^{\infty} r(t-\tau)w(\tau)\,d\tau = (r \bullet w)(t)\,, \tag{5.6}$$

where $r(t)$ is the *reflectivity series* and $w(t)$ is the wavelet as before. Though superficially similar to Eq. (5.5), the convolutional model is actually very different because the reflectivity series is quite distinct from the impulse response. As discussed previously, the impulse response contains all possible physical effects that arise from a point source. This includes primary (single-bounce) reflections, all possible multiples, mode-converted waves, anelastic attenuation, spherical spreading, and all other effects found in seismic waves. In comparison, the reflectivity function is defined as a function of time and/or space (for simplicity, we will usually neglect the latter) whose value at any point is the Earth's reflection coefficient. This is a very nonunique definition, since reflection coefficients are known to be functions of incidence angle and also to depend upon the wave mode for both incident and reflected waves. That is, we may have the reflection coefficient for P-waves reflecting as P-waves, or for P-waves reflecting as S-waves, etc. Again, for simplicity and for the sake of uniqueness, we will assume that we are speaking of P-waves reflecting as P-waves at normal incidence. With this assumption, we can regard the reflectivity function as a directly measurable quantity at a well, although it is measured in depth and we need to know its value in time.

By using Eq. (5.6) rather than Eq. (5.5), the deconvolution problem is greatly simplified but in a sense we have defaulted on a very important issue. Assuming that Eq. (5.5) is a good model for raw data, how are we to prepare the data for deconvolution such that Eq. (5.6) is a reasonable approximation? The central issue is to understand how the impulse response and the reflectivity are related. In 1D, where there is no mode conversion or wavefront spreading, the difference between impulse response and reflectivity is illustrated clearly in Figure 4.23a, where the trace labeled "primaries and multiples" is the impulse response. Figure 5.1 redisplays the reflectivity and impulse response from the former figure with the impulse response reversed in polarity to better match the reflectivity. Also shown is the result of applying a $Q = 80$ Q matrix to both. There are two essential differences between reflectivity and impulse response in 1D, and these are the transmission losses and the presence of multiples in the latter. The lossless traces are an idealization, and more realism comes from the application of the Q matrix to simulate anelastic attenuation. A first-order extension of these results to 3D would be to apply the amplitude decay expected from wavefront spreading, which would further diminish the signals.

The most obvious thing that should be done to prepare data for deconvolution is some sort of gain or amplitude correction, as will be discussed in the next section. Ultimately, the convolutional model, as expressed by Eq. (5.6), is only a very coarse approximation to the real complexity of a seismic record. The real virtue of this model is that it allows us to state the deconvolution problem in a simple, solvable manner. Understanding the strengths and weaknesses of the resulting algorithm will come later.

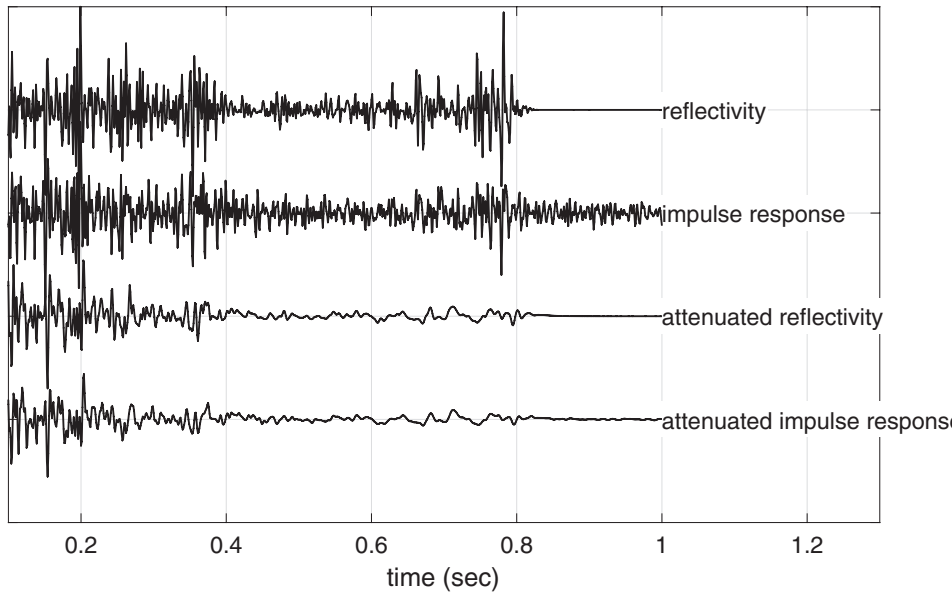


Figure 5.1 The reflectivity and impulse response are compared for a 1D seismogram. This is the same as shown in Figure 4.23a except that the impulse response has been reversed in sign. Also shown are the results of applying a $Q = 80$ Q matrix to both.

5.2 Gain Correction

Gain correction refers to techniques designed to remove the amplitude decay seen in raw seismic traces, thus hopefully moving the traces more closely to the convolutional model. The main physical causes of this amplitude decay are (i) wavefront spreading, (ii) transmission losses, and (iii) anelastic attenuation. The first two effects depend only on time and are independent of frequency; however, as we have seen, attenuation is both time and frequency dependent. Thus we should expect attenuation will be more challenging to address.

5.2.1 Time-Dependent Gain

Consider a spherical wavefront expanding from a point source in a homogeneous medium. At time t_1 since source initiation, the wavefront will have a radius $r_1 = t_1 v$, where v is the constant velocity. Similarly, the radius at a later time t_2 is $r_2 = t_2 v$. Conservation of energy requires that the energy on these two surfaces must be equal, so

$$E_1 = 4\pi r_1^2 e_1 = E_2 = 4\pi r_2^2 e_2, \tag{5.7}$$

where e_1 and e_2 are local "energy densities" on the two surfaces. A geophone placed on either surface will measure the local particle velocity, which is proportional to the square

root of the local energy density.[1] As a result, we expect the geophone amplitudes at the two times to be related by

$$A_2 = \frac{A_1 t_1}{t_2}. \tag{5.8}$$

Since we do not know the initial amplitude at some reference time, this is usually generalized to

$$A(t) = \frac{c}{t}, \tag{5.9}$$

where c is some unknown constant. Given a raw seismic trace, a gain correction for spherical spreading could then take the form

$$s_g(t) = s(t)t, \tag{5.10}$$

meaning that we simply multiply each sample of the trace by its corresponding time. Although this expression has been developed for the extremely simple and unphysical case of a homogeneous medium, note that we have essentially admitted to amplitude restoration that is in error by an unknown constant scale factor. This circumstance will be present all through deconvolution theory and will only get worse in real, complicated media. Equation (5.10) considers only wavefront spreading and ignores the additional decay from transmission losses and attenuation. Also, in variable-velocity media, we can expect this result to be only very approximate. This leads to the common generalization that is often employed in practice, called *t-gain*,

$$s_g(t) = s(t)t^n, \tag{5.11}$$

where n is a number to be determined by trial and error and is usually found to be between 1 and 2. In order to estimate n, it is usually necessary to have some idea of the expected amplitude behavior after correction. Most commonly, it is assumed that some kind of average amplitude measure taken over a chosen window size should be nearly constant. Specifically, we might compute the rms amplitude over perhaps a 0.3 s moving window and require this to be roughly constant after t-gain. The function `tgain` applies Eq. (5.11) once n has been determined. The t-gain method with an empirically determined n is an example of a statistical model being applied to the underlying reflectivity.

Deterministic correction for transmission loss would require a knowledge of stratigraphy that is usually not available. Similarly, it is very difficult to do a more accurate correction for wavefront spreading without additional information. At present, the most accurate wavefront-spreading corrections are done in prestack depth migration after determining an accurate velocity model.

5.2.2 Automatic Gain Correction

Another simple method of gain correction comes under the general appellation of *automatic gain control*, or AGC. There are many possible AGC methods, though all have

[1] The local energy density will be proportional to the kinetic energy associated with particle motion, which is proportional to velocity squared.

roughly similar behavior. The basic idea is to assume that the trace amplitudes should be roughly constant over time, similarly to the t-gain method for determining the exponent n. This assumption, in turn, implies that the underlying reflectivity, $r(t)$, is also roughly constant over time. Thus, if the uncorrected trace shows systematic amplitude decay, as almost all raw traces do, then we need to estimate that decay and somehow remove it.

To give some precision to these rather vague statements, we now examine the algorithm for a form of AGC called *automatic envelope correction*, or AEC. In Section 2.4.6, the idea of a Hilbert envelope (see Eq. (2.103)) was introduced and it was argued that the envelope provides a phase-independent measure of the amplitude behavior. The Hilbert envelope can be used as an amplitude measure for an AGC process, but it must first be smoothed so that local amplitude variation is preserved. This requires the choice of a temporal-smoother length, commonly called the AGC length. Following the notation in Section 2.4.6, the envelope of a trace $s(t)$ is denoted by $\epsilon_s(t)$, and its smoothed representation by $\overline{\epsilon_s}(t)$. Then the AEC corrected trace is given by

$$s_{\mathrm{g}}(t) = \frac{s(t)}{\overline{\epsilon_s}(t) + \mu\epsilon_{\max}}, \tag{5.12}$$

where $0 < \mu < 1$ and $\epsilon_{\max} = \max \overline{\epsilon_s}$. The parameter μ is called a stability constant and, technically, is there to prevent division by zero should the smoothed envelope ever become zero. However, this is unlikely and, more practically, this prevents overamplification of small amplitudes, which often occur at the beginning and end of a trace. Code Snippet 5.2.1 shows the computation of Eq. (5.12) and Figure 5.2, illustrating a trace before and after AEC correction, and the Hilbert envelope and its smoothed self. The most important parameter is the operator length, because this controls the relative amplitude behavior of the result. Reflection events whose time separation is much less than the operator length can still have their amplitudes meaningfully compared, while events with greater separation may not be (see Exercise 5.2.1). Another effect of the operator length that is often of concern is that random noise appearing before the first break will be amplified if its time separation from the first break is greater than the operator length (see Exercise 5.2.2).

Exercises

5.2.1 Repeat the calculations of Code Snippet 5.2.1 for operator lengths of 0.02, 0.05, 0.1, 0.2, 0.4, and 0.8, and make a plot comparing the results. Describe the effects of varying the operator length. Which length do you prefer and why?

5.2.2 Repeat the calculations of Code Snippet 5.2.1 but change the input trace by attaching 0.5 s of weak random noise to the front of the trace. You can use `rnoise` for this and choose a signal-to-noise ratio of 10. Then apply the AEC method using an operator length of 0.2 s. Make a plot of your results and explain them.

5.2.3 Repeat the calculations of Code Snippet 5.2.1 using a set of different stability constants. Examine the values 0.1, 0.01, 0.001, and 0.0001. Make a plot comparing your results. Describe the effects of varying the stability constant. Which value do you prefer and why?

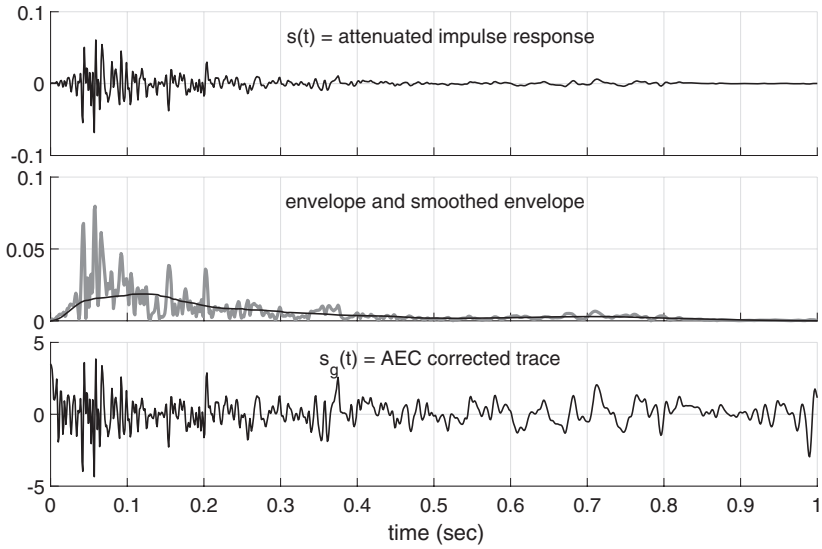


Figure 5.2 Automatic gain correction by the AEC method. At top is the trace to be corrected, which is the "attenuated impulse response" of Figure 5.1. In the middle is the Hilbert envelope and its smoothed representation using a convolutional smoother of length 0.2 s. At bottom is the result of dividing the attenuated impulse response by its smoothed envelope, as shown in Code Snippet 5.2.1.

Code Snippet 5.2.1 This illustrates the basic steps for AGC by the method of automatic envelope correction, or AEC. Line 1 computes the Hilbert envelope of a trace `s`, which is the "attenuated impulse response" of Figure 5.1. Line 2 defines the smoother length in seconds and line 3 computes the length in samples. Line 4 defines a stability constant, used to stabilize the division. Line 5 smooths the Hilbert envelope using convolution with a boxcar (note the division by `nsmo`). Finally, line 7 divides the trace `s` by its smoothed envelope, using the stability constant to avoid overemphasizing small amplitudes. This process is illustrated in Figure 5.2.

```
1  e=env(s);%compute the Hilbert envelope
2  tsmo=.2;%define the smoother length in seconds
3  nsmo=round(tsmo/dt);%smoother length in samples
4  stab=.01;%define stability constant
5  esmo=convz(e,ones(nsmo,1))/nsmo;%compute smoothed envelope
6  emax=max(esmo);%maximum value of the smoothed envelope
7  sg=s./(esmo+stab*emax);%stabilized division for AEC
```

End Code

deconcode/aec_method.m

5.2.3 Discussion

The topic of gain, or more generally *amplitude restoration*, gets a lot of attention in the seismic community. This is probably because seismic images are the best-known way to

interrogate the subsurface and interpreters like to have faith in the amplitudes that they see. For example, if a particular horizon shows a local 50% amplitude increase, then they would like to reliably infer that the reflection coefficient has therefore increased by 50%. The two gain methods discussed here are examples of what are called deterministic (t-gain) and statistical (AGC) methods. There are many variations of these two basic approaches, but the statistical methods are those which base their gain correction on the data, and they usually employ a "reflectivity model" either explicitly or implicitly. With AGC, the reflectivity model is that the reflectivity is essentially constant when averaged in time over lengths greater than the operator length. At the other extreme are the deterministic methods, which base their corrections on a formula thought to model the expected amplitude variations that are not due to reflectivity. Removing these variations via the formula should then reveal the reflectivity. However, just as with t-gain, all deterministic methods require an element of data fitting and parameter selection and so are not truly independent of the data or a model.

Also entering into the discussion is the ill-defined phrase "true-amplitude processing." Taken literally, this implies that a data-processing flow is capable of producing a seismic image whose amplitudes are truly reflectivity. At present, and for the foreseeable future, this is essentially impossible and must be regarded more as a goal or philosophy than as a real possibility. To truly realize reflection coefficients from data processing means that all amplitude effects that are not reflectivity must somehow be detected and removed. The list of effects is long and includes wavefront spreading, transmission losses (including mode conversions), attenuation effects, source waveform, source strength, geophone response, near-surface complications, and more. Most of these effects depend upon properties of the Earth that are highly variable with position and are also closely related to the reflectivity itself. The implication is that true-amplitude processing requires almost complete knowledge of subsurface properties, and this is clearly not possible. The only emerging technology that may offer a solution is full-waveform inversion, but this is far from practical and requires a computational ability that is orders of magnitude greater than that currently available. What is more possible is "relative-amplitude processing." This implies that the data is processed such that the ratio of any two amplitudes is the same as the ratio of their underlying reflection coefficients. Even this is not possible without caveats, the most important being that the amplitudes being compared are from similar stratigraphic positions and reasonably close together spatially. Meeting even this goal requires very sophisticated algorithms and great attention to detail and quality control during the data-processing sequence. It is sometimes claimed that all that is required is to avoid AGC in favor of more deterministic methods. However, such a simple prescription leads to many problems, because AGC is a very powerful tool and, while it can distort amplitudes, it can also lead to higher-frequency images. One of the most useful and robust processes in relative-amplitude processing is the surface-consistent adaptation of gain and of deconvolution. Called surface-consistent amplitude restoration and surface-consistent deconvolution, such methods develop corrections and operators that are functions of source and receiver position (and also midpoint and offset) rather than allowing trace-by-trace variation. The surface-consistent approach will not be developed in this book but is well explained elsewhere, including in Yilmaz (2001).

So, if true-amplitude processing is impossible, how then can seismic images be reliably interpreted and successful wells drilled based on those images? The answer lies in work done within energy companies after the data processing is complete. Called *well-tying*, this involves comparing the final seismic image (usually a 3D volume) with synthetic seismograms created from well control at locations that fall within the volume. From these comparisons, it is hoped to deduce the residual embedded wavelet(s) remaining in the data and somehow compensate for these. Here the term "embedded wavelet" usually means whatever time series needs to be convolved with the well reflectivity in order to match the seismic data at the well. Often the match is only assessed qualitatively, but an attempt is then made to phase-rotate the volume to a "zero-phase state," meaning that the embedded wavelet should then be zero phase. Spatial variations in amplitude between wells are only sometimes addressed.

5.3 Frequency-Domain Stationary Spiking Deconvolution

There are two main algorithms for what is called stationary spiking deconvolution: the frequency-domain and time-domain algorithms. Both are encapsulated in functions in the *NMES Toolbox* (`deconf` and `deconw`) and, with intelligent parameter selection, it is possible to get essentially identical results from both. However, the frequency-domain method will be discussed first because it is conceptually easier to grasp, while the time-domain method is dominant in industry for mainly historical reasons. Both methods are based strictly on the convolutional model, assume that the unknown wavelet is minimum phase, and assume that the reflectivity is "white." The word *stationary* in this context means that temporal evolution of the seismic wavelet is not modeled and that the designed deconvolution operator, which is an approximate inverse of the estimated wavelet, is applied to the entire trace. While this explicit denial of nonstationary attenuation processes, which are always present, may seem a serious problem, there are coping strategies which allow some increased flexibility. The word *spiking* here refers to the intent of the (deconvolution) operator design, which is to collapse the estimated wavelet into an approximate spike. Thus the operator is an approximate inverse of the estimated wavelet. Since the wavelet is assumed to be minimum phase, this spiking can only succeed if that assumption is met.

A further assumption is that the reflectivity should be *white*. This jargon comes from optics, where white light is characterized by having equal power at all frequencies. Figure 5.3 compares a real well-log-derived reflectivity with a computer-generated random reflectivity from the function `reflec`. The real reflectivity was generated from sonic and density logs by the `seismo` command (see Code Snippet 4.7.11) with a time sample size of 0.0005 s. The random reflectivity comes from `reflec` with the same sample rate and length. While the time-domain appearances are quite distinct, it is in the frequency domain where the significance for deconvolution is most apparent. The random reflectivity has a spectrum that, if smoothed with a smoother of a certain length (perhaps 10 Hz), is essentially flat or white. The real reflectivity, on the other hand, shows a distinct rolloff, beginning at around 100 Hz, to lower power at lower frequencies. If a white spectrum is

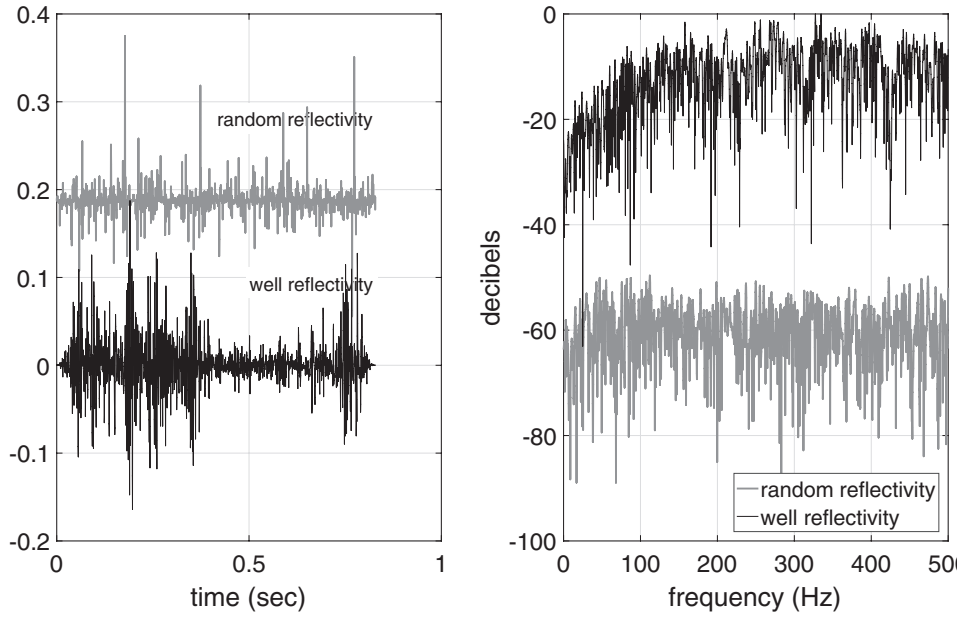


Figure 5.3 The real reflectivity from the same well as shown in Figure 4.23a is compared with a computed random reflectivity generated by the `reflec` function. On the left is the time domain, and on the right is the frequency domain. The spectrum of the random reflectivity has been bulk-shifted down for better viewing. Both time series are essentially zero mean, but the random reflectivity has been bulk-shifted up.

taken as an indication of randomness, then the real reflectivity shows departure from randomness at lower frequencies. The real reflectivity comes from the sequence of layering laid down by geological processes. It follows that such processes are random at the finer scales (perhaps thousands of years) but are somehow correlated at longer timescales (millions of years). What is significant is that real reflectivities do not fit the "white" model, and this becomes more significant if the seismic bandwidth is very broad. This nonrandom behavior is found all around the Earth.

For the purpose of exposition, the random reflectivity will now be the focus. The central problem of deconvolution is not the creation of the inverse operator, but the estimation of the wavelet that must be inverted. This wavelet is almost always unknown, and the success or failure of deconvolution rests largely on the quality of the wavelet estimation. From a mathematical perspective, this seems essentially impossible, but it is the role of the assumptions of white reflectivity and minimum phase to turn this into a possibility. Consider a trace $s(t)$ formed from the convolution of the reflectivity and wavelet as $s(t) = (r \bullet w)\,(t)$. In the frequency domain, this is simple multiplication, $\hat{s}(f) = \hat{r}(f)\hat{w}(f)$, which must be independently true at each frequency. Since only $\hat{s}(f)$ is known, we are trying to separate one complex number into the product of two complex numbers. Without further information, there are infinitely many possibilities for this separation. Figure 5.4 suggests the way forward, which is to observe that in the frequency domain, the amplitude spectrum of the wavelet can be deduced by smoothing that of the seismic trace. For the noise-free

trace, this is true at all frequencies, while for the noisy trace this is only so when signal dominates noise, which in this case is below 60 Hz. This relationship also relies critically upon the reflectivity spectrum being white, or shapeless, when smoothed. Thus, a possible strategy is to Fourier transform the trace and smooth its amplitude spectrum to estimate the wavelet's amplitude spectrum. However, this relationship is not true for the phase spectra, so something else must be done, and this is the assumption of minimum phase. If the wavelet is minimum phase, then its phase spectrum can be computed from the amplitude spectrum by $\phi_w(\omega) = -\mathcal{H}[\ln(A_w(\omega))]$ (see Section 2.4.8). Finally, given the wavelet estimate, we deduce its frequency-domain inverse by simple division as the spectrum of the deconvolution operator.

Essential algorithmic flexibility is gained by distinguishing between the trace used to design the operator and the trace to which the operator is to be applied. For example, it is often desired to restrict the operator design to a limited portion of a trace which spans a target interval and then apply the operator to the entire trace. This can be considered as some accommodation toward the essential nonstationarity of seismic traces. If the entire trace is used for operator design, then the estimated wavelet will only be some sort of average of the evolving wavelet and the target reservoir may be left underresolved. By focusing the design on the interval of most interest, the wavelet can be more optimal for that interval at the cost of introducing over correction or under correction elsewhere. Another application is that an ensemble of traces can all be deconvolved with the same operator, which

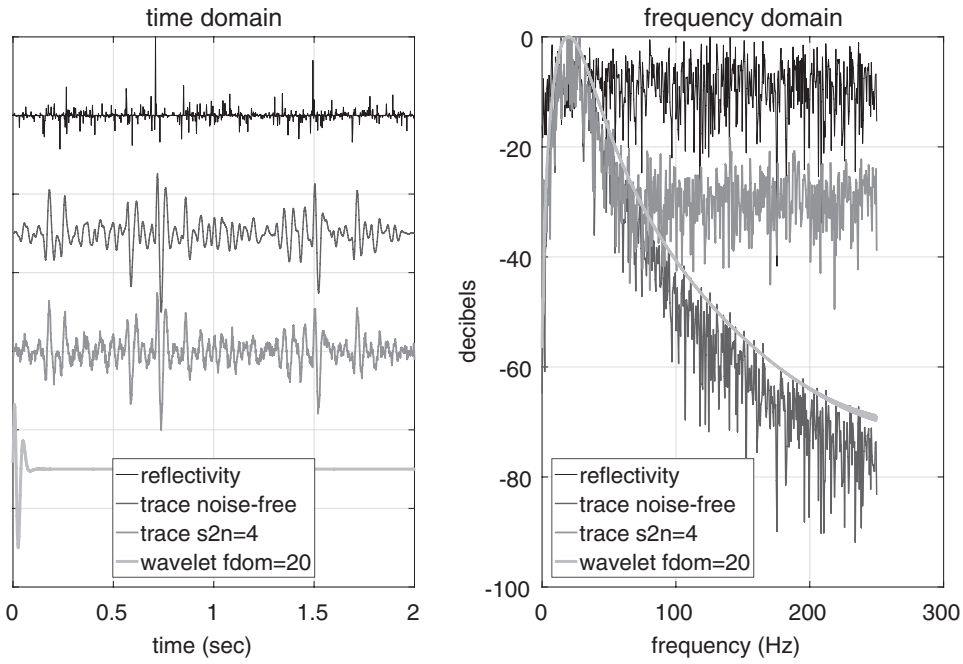


Figure 5.4 Synthetic convolutional seismograms with and without noise are shown together with the reflectivity and the wavelet. At left is the time domain and at right is the frequency domain (amplitude spectra).

may be designed from an average trace. Code Snippet 5.3.1 illustrates the essential steps in frequency-domain deconvolution, as extracted from *deconf*. For the corresponding mathematical description, let $u(t)$ be the design trace and $s(t)$ be the trace to be deconvolved. Then the first step of operator design is to compute the power spectrum of the design trace as

$$P(f) = \left|\hat{u}(f)\right|^2. \tag{5.13}$$

The entire algorithm could be based on the amplitude spectrum rather than the power spectrum, but we choose the latter in order to maintain better equivalence with the time-domain algorithm to be discussed later.

Next, a small positive constant is added to the power spectrum in order to prevent possible division by zero or a very small number and also to avoid influence from noise. If the noisy trace of Figure 5.4 is to be deconvolved, then the ideal value for this small number corresponds to a power level just above that of the noise, which swamps the signal at about 60 Hz. With real data, this can be very difficult to determine, but the concept is clear. Commonly, this small additive power is expressed as $\mu P_{\max}$, where $P_{\max} = \max P(f)$ and $0 < \mu < 1$. Typically μ is between 0.0001 and 0.1, with larger values being preferred for higher levels of noise. This being done, the modified power spectrum is then smoothed by a convolutional process

$$\bar{P}(f) = [P(f) + \mu P_{\max}] \bullet b(f), \tag{5.14}$$

where $b(f)$ is a suitably normalized "bump function," meaning a function centered about $f = 0$ with a definable width and whose value is never negative. *deconf* allows the choice of a boxcar, triangle, or Gaussian as a bump function, although this is not a critical option. The choice of the value of μ and the width of the bump function are the more important considerations. This computation happens on lines 6–9 in Code Snippet 5.3.1, where it is important to note the role of `fftshift` to create the symmetric form of the spectrum before smoothing. This form allows proper convolutional smoothing across $f = 0$. The smoothing lightly distorts the perfect symmetry of the power spectrum, so lines 10 and 11 restore this.

Roughly speaking, the bump function width, δf_b, should have a value between about 2 and 20 Hz. Examine again Figure 5.4 and note the essential difference between $\left|\hat{s}\right|$, the amplitude spectrum of the trace, and $\left|\hat{w}\right|$, the amplitude spectrum of the wavelet. They have the same general shape, but $\left|\hat{s}\right|$ is full of sharp, rapid oscillations that are characteristic of the reflectivity. The purpose of the smoothing is to suppress the reflectivity contribution to $\left|\hat{s}\right|$, with the idea that what remains is the wavelet. A number of things are clear about this smoothing, including (i) it is not a unique process; there are many ways to do it; (ii) the smoother must not be too long or too short; and (iii) the process of smoothing amounts to an algorithmic bias. The nonuniqueness arises not just because there are infinitely many bump-function shapes but also because the separation of $\left|\hat{s}\right|$ into a smooth part and a rugged part could be done in many other ways. Ulrych (1971), for example, proposed an elegant filtering process, called homomorphic deconvolution, that has many attractive features but will not be examined here. In point (ii), clearly a very long smoother gives the wrong answer because, in the limit as the smoother length increases without bound, the result tends to a constant, clearly the wrong answer. Similarly, in the limit as the smoother length

Code Snippet 5.3.1 The key steps of frequency-domain deconvolution are illustrated. The input trace (to be deconvolved) is called `trin` and the design trace (from which the operator is estimated) is called `trdsign`. The length of the spectral smoother in samples is `nn`, which has been previously forced to be an odd number. On line 1, the design trace is padded with zeros to the same length as the input trace. The power spectrum of the design trace is computed on lines 3 and 4. Note the use of `fftshift` to create a symmetric spectrum with $f = 0$ in the middle. This is essential for proper smoothing. Line 6 adds the background white noise power for stability, and lines 8 and 9 smooth the power spectrum by convolution with a boxcar. The convolutional smoothing slightly alters the perfect symmetry of the power, so lines 10 and 11 restore this. Lines 13 and 15 compute the spectrum of the minimum-phase inverse operator. `hilbert` is used to accomplish the Hilbert transform. Finally, lines 17–19 accomplish the deconvolution of `trin` by `fft`, multiplication, and `ifft`. This code is excerpted from *deconf*.

```
1  trdsign=pad_trace(trdsign,trin); %pad trdsign to length of trin
2  % generate the power spectrum
3  spec= fftshift(fft(trdsign));%note fftshift
4  power= real(spec).^2 + imag(spec).^2;
5  % stabilize the power spectrum
6  power=power+stab*max(power);
7  % smooth the power
8  smoo=ones(nn,1);%nn is an odd number of samples
9  power=convz(power,smoo,ceil(nn/2),length(power),0)/sum(abs(smoo));
10 n2=length(power);
11 power(n2/2+2:end)=power(n2/2:-1:2);%enforce symmetry
12 % compute the minimum phase spectrum
13 logspec=hilbert(.5*log(power));% .5 because power not amplitude
14 % compute the complex spectrum of the inverse operator
15 specinv= exp(-conj(logspec));%- sign for inverse
16 % deconvolve the input trace
17 specin=fftshift(fft(trin));%note fftshift
18 specout=specin.*specinv;%decon is just multiplication
19 trout=real(ifft(fftshift(specout)));%note fftshift and real
```

End Code

deconcode/deconf_guts.m

decreases toward zero, the result is an unchanged spectrum, also clearly wrong. Finally, in point (iii), the process is said to be biased because if it is given a perfectly simple input it will always get the answer slightly wrong. Suppose the reflectivity is just a single impulse in the middle of a string of zeros. Then $|\hat{s}|$ will equal $|\hat{w}|$ exactly, but if any convolutional smoothing is done the estimate will be slightly wrong. (The homomorphic approach can be shown to avoid this bias.)

Now, given $\bar{P}(f)$, the next step is to compute the corresponding minimum-phase spectrum and then form the spectrum of the deconvolution operator, $\hat{d}(f)$. First, since the power spectrum is the square of the amplitude spectrum, it follows that

$$\ln \left|\hat{d}\right| = \ln \bar{P}^{-1/2} = -0.5 \ln \bar{P}, \tag{5.15}$$

and then from Eq. (2.126) we have for the phase of $\hat{d}(f)$

$$\phi_d(f) = 0.5\mathcal{H}\left[\ln \bar{P}\right], \tag{5.16}$$

where $\mathcal{H}[\cdot]$ denotes the Hilbert transform. The equivalent computational step is line 13 of Code Snippet 5.3.1, where `hilbert` performs the Hilbert transform, which takes a real-valued vector as input and returns a complex-valued vector whose real part is the input and whose imaginary part is the Hilbert transform of the input. For this reason, line 15 computes the entire spectrum of the operator, not just the phase

$$\hat{d}(f) = \left(\bar{P}^{-1/2} e^{0.5\mathcal{H}[\ln \bar{P}]}\right)(f). \tag{5.17}$$

Written in this way, the expression emphasizes that the amplitude and phase of the deconvolution operator are both determined by the smoothed, stabilized power spectrum $\bar{P}(f)$. No phase information is involved in determining the operator, so this means that the phase of the input data has no effect on the computation of the operator. This can be a confusing point, since it is always stated that the input data must be minimum phase before deconvolution.[2] For better understanding, first consider the application of the operator to the trace $s(t)$ to produce $s_d(t)$

$$\hat{s}_d(f) = \hat{s}(f)\hat{d}(f) = \hat{r}(f)\hat{w}(f)\hat{d}(f) = \hat{r}(f)\hat{w}_d(f), \tag{5.18}$$

where the third expression comes from the substitution $\hat{s}(f) = \hat{r}(f)\hat{w}(f)$ and, in the final expression, $\hat{w}_d(f) = \hat{w}(f)\hat{d}(f)$ is the spectrum of the embedded wavelet after deconvolution. This step happens on line 18 of Code Snippet 5.3.1, and the transformation back to the time domain is on line 19. The use of `real` on line 19 is necessitated because, despite all efforts to preserve the correct symmetry for a real-valued signal (see Section 2.4.2), a direct (ifft) produces a tiny imaginary part owing to numerical loss of precision. A perfect deconvolution will have the result $\hat{s}_d(f) = \hat{r}(f)$, meaning that $\hat{w}_d(f) = 1$ so that $w_d(t)$ is a perfect Dirac delta spike. We do not expect perfection, but we hope for $w_d(t)$ to be an approximate spike, and this will only be the case if the phase of $\hat{w}$ and the phase of $\hat{d}$ sum to nearly zero. In turn, this can only happen if $\hat{w}$ is minimum phase. Suppose an all-pass filter with spectrum $1e^{i\phi(f)}$ were applied to both $u(t)$ and $s(t)$ before deconvolution. This will not affect the operator design, and so it follows that the embedded wavelet after deconvolution will now be $\hat{w}_d(f) = \hat{w}(f)\hat{d}(f)e^{i\phi(f)}$. The operator $d(t)$ is minimum phase by design and, even if $w(t)$ is also minimum phase, $w_d(t)$ cannot be unless $\phi(f) = 0$. This is not a desirable result, because $w_d(t)$ might be very noncompact and distorted, and a further deconvolution will not improve things.[3]

Frequency-domain deconvolution is a good example of a seismic algorithm where it is crucial to get the phase calculations done correctly. It can be very confusing to be sure that a developed code has the correct sign on the phase term, and this is especially so since

[2] By which it is meant that the embedded wavelet must be minimum phase, not the trace itself.

[3] It is desirable that the output from deconvolution be minimum phase so that a subsequent deconvolution is possible if it is judged that the data are underresolved.

Code Snippet 5.3.2 Here is an example of a simple test to illustrate both the best possible result from deconvolution and also what happens if the operator phase has a sign error. Lines 1–7 build a synthetic trace which has a minimum-phase wavelet convolved with a reflectivity with a single unit spike (created by the *impulse* command on line 4). Lines 8–10 define the essential parameters for *deconf*, which are the number of samples in the smoother, n, and the stability constant, stab. Line 11 runs *deconf*, where the first input is the trace to be deconvolved and the second input is the design trace. The first output is the deconvolved trace and the second is the spectrum of the deconvolution operator. Finally, line 13 simulates what happens if the operator phase has a sign error by taking the complex conjugate of the operator and applying it to the input trace. The results are shown in Figure 5.5a.

```
1   dt=.002;%time sample size
2   t=dt*(0:511)';%t is 512 samples long
3   tmax=t(end);%max time
4   r=impulse(t);%a single impulse in the middle
5   fdom=20;%dominant frequency
6   [w,tw]=wavemin(dt,fdom,tmax/2);%min phase wavelet
7   s=convm(r,w);%trace
8   delf=5;%deconf smoother in Hz
9   n=round(delf*tmax);%deconf smoother in samples
10  stab=0.00001;%mu the stability constant
11  [sd,specd]=deconf(s,s,n,stab);%specd is the inverse op spectrum
12  %2nd result with op phase flipped
13  sd2=real(ifft(fft(s).*fftshift(conj(specd))));
```

End Code

deconcode/ phserror . m

the sign convention used in the Fourier transform software must also be considered. A code should therefore always be tested on synthetic data built for the purpose of testing the phase. For example, a synthetic trace where the wavelet is minimum phase and the reflectivity is a single spike in the middle of the trace is a good choice. Since a sign error in the phase causes time reversal, then if the sign of the phase of the operator is correct, then the deconvolved result will have a slight causal tail, whereas if the sign is incorrect, the tail will be anticausal. Code Snippet 5.3.2 is an example of such a test. The trace to be deconvolved has a single isolated minimum-phase wavelet, and so we might hope that the result would be a perfect spike. Figure 5.5a shows the result, labeled "deconvolved," and it is visually clear that the result, while very good, is not quite a perfect spike. This is the embedded wavelet after deconvolution, $w_d(t) = (w \bullet d)(t)$ or, in the frequency domain, $\hat{w}_d(f) = \hat{w}(f)\hat{d}(f)$. Again, since $d(t)$ is always minimum phase by construction and since $w(t)$ is also minimum phase, then $w_d(t)$ is minimum phase. The deconvolution is imperfect in this case because the stability constant was nonzero and because smoothing always slightly distorts the spectrum. In any practical setting, the deconvolution is also imperfect but for far more serious reasons: the data is nonstationary, the real reflectivity is colored, noise is present, and so on. Even then, with real data, if the input data is minimum phase

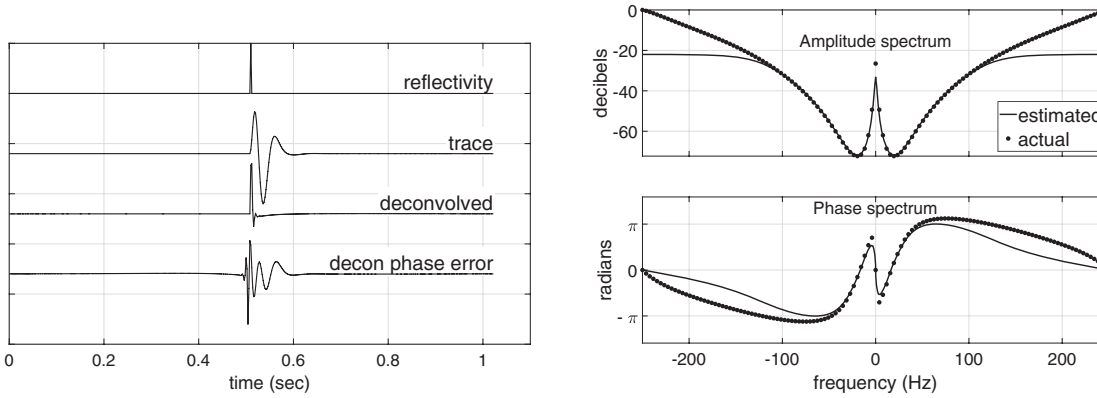


Figure 5.5a (left) The results from Code Snippet 5.3.2. The result labeled "deconvolved" is `sd` and is an example of an almost perfect deconvolution. The embedded wavelet is nearly a spike but still obviously causal and, less obviously, minimum phase. The result labeled "decon phase error" is `sd2` and illustrates what happens if the phase of the operator design has a sign error. The anticausal tail extending before the reflection spike is the primary indicator.

Figure 5.5b (right) The amplitude and phase spectra of the deconvolution operator, `specd`, from Code Snippet 5.3.2, compared with the actual spectra of the inverse wavelet.

(i.e., the embedded wavelet is minimum phase), then the data will still be minimum phase after deconvolution. This means that a second pass of deconvolution, perhaps after noise reduction or after stacking, will almost always improve the results. Figure 5.5b shows the amplitude and phase spectra of the deconvolution operator from Code Snippet 5.3.2 in comparison with those of the actual inverse wavelet. The time sample size is $\Delta t = 0.002$ s, so the Nyquist frequency is 250 Hz. The amplitude spectrum is quite well estimated out to about half-Nyquist, which is an exceptionally good result. At higher frequencies, the spectrum flattens out owing to the additive white noise term $\mu P_{\max}$ (Eq. (5.14)). The sharp turn upward near Nyquist is an edge effect of the convolutional smoother. The phase spectra show an error that seems significant below half-Nyquist, and this is because the Hilbert transform is a nonlocal integration that spreads the errors in the amplitude spectrum around. The phase of $w_d(t)$ is $\phi_w + \phi_d$, while the phase of the inverse wavelet is $-\phi_w$. Therefore, the difference of the two phase curves in Figure 5.5b is the phase of the postdeconvolution wavelet. This is often called the *residual phase*; it is generally not constant and is larger at higher frequencies.

Exercises

5.3.1 Repeat the computation in Code Snippet 5.3.2, ignoring the calculation in line 12, for several different `stab` values both larger and smaller and present your results similarly to Figures 5.5a and 5.5b. Describe the influence of `stab` on the embedded wavelet and the operator spectra.

5.3.2 Repeat the computation in Code Snippet 5.3.2, ignoring the calculation in line 12, for several different frequency smoothers `delf`. Use values both larger and smaller and present your results similarly to Figures 5.5a and 5.5b. Describe the influence of `delf` on the embedded wavelet and the operator spectra.

5.3.3 Explain why line 9 in Code Snippet 5.3.2 is the correct way to calculate the smoother width in samples. That is, if a smoother has width δf_b in hertz, why is the equivalent number of samples given by $n = \delta f_b t_{\max}$, where $t_{\max}$ is the trace length in seconds?

5.3.1 A Test of Frequency-Domain Deconvolution on a Stationary Synthetic

As an illustration of the use and performance of *deconf*, consider the deconvolution of the noiseless and noisy synthetic seismograms of Figure 5.4. For this purpose, we must specify the frequency-domain smoother length and type and the `stab` values for both seismograms. The smoother length and type can be the same for both, and for this example let the length be 10 Hz and the type be Gaussian. However, usually the `stab` values will be chosen differently depending on the perceived noise levels. For real data, this is quite difficult and subjective, but for this synthetic data, we can see from the figure that, for the noisy trace, noise swamps signal at about 60 Hz and this is about 30 dB down. Recall from Eq. (5.14) that `stab` is symbolized by μ and determines a power level that will be added to the actual power spectrum. Chosen properly, this can be used to suppress the spectral whitening action of the deconvolution. For the noisy trace, if $\mu P_{\max}$ is chosen to correspond to a level just above the noise, this will have the desired effect. Equation (1.1) defines decibel levels for an amplitude spectrum. When working with a power spectrum, the equivalent formula is

$$P_{\mathrm{dB}}(f) = 10 \log_{10} \left(\frac{P(f)}{P_{\mathrm{ref}}} \right), \tag{5.19}$$

where the leading 20 has been replaced by 10 because $P = A^2$. Using $P_{\mathrm{ref}} = P_{\max}$, we find that μ values of [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001] correspond to $[-10, -20, -30, -40, -50, -60]$ dB. Therefore a `stab` of 0.01 seems reasonable for the noisy trace and, somewhat arbitrarily, we choose 0.000001 for the noise-free trace.

Code Snippet 5.3.3 performs the deconvolution of both traces, and the results are displayed in Figure 5.6 for the time domain and Figure 5.7 for the frequency domain. A significant new feature of this code is the use of *maxcorr_phs* to measure the similarity of the reflectivity and the deconvolved trace. This is an extension of *maxcorr* (see Section 2.2) that also measures the best constant-phase rotation required to match two signals in the least-squares sense. The constant-phase estimation is contained in either *constphase* or *constphase2*. Suppose we have two signals s_1 and s_2; then these codes estimate the phase angle θ_0 that minimizes $\sum (s_2 - s_{1\theta})^2$, where the sum is over samples; $s_{1\theta}$ is a phase rotation of s_1 in the sense of Eq. (2.95). There is an analytic solution to this problem (not presented here) that is exploited by *constphase*, while *constphase2* takes

Code Snippet 5.3.3 Here is an example of a code to test *deconf* on the two traces of Figure 5.4. The noise-free trace is `s`, the noisy trace `sn`, and the reflectivity `r`. Line 1 defines the smoother length in hertz and the `stab` and `stabn` values, where `n` denotes the noisy case throughout this snippet. On line 2, the smoother length in samples is calculated. Lines 3 and 4 define more *deconf* parameters. Line 5 specifies band-pass filter parameters for a postdeconvolution filter. Line 6 specifies the number of crosscorrelation lags to search in comparing the deconvolved traces with the reflectivity. Line 7 deconvolves the noise-free trace, where the trace and the design trace are the same. On line 8, the function *maxcorr_phs* is used to compare the deconvolved trace with the reflectivity. The returned values are the maximum crosscorrelation coefficient, the lag in samples at which this occurs, and the best constant-phase rotation required to match the deconvolved trace to the reflectivity. The second return, `str`, is a string containing these values for easy annotation on a plot. Lines 9 and 10 apply a 5–125 Hz band-pass filter to both the reflectivity and the deconvolved trace, and the comparison is repeated after filtering on line 11. Then, lines 12–16 repeat this process for the noisy trace using different parameters where necessary. The results are shown in Figures 5.6 and 5.7.

```
1  fsmo=10;stab=.000001;stabn=.01;%deconf parameters
2  nsmo=round(fsmo*max(t));%smoother length in samples
3  stype='gaussian';%smoother type
4  phase=1;%decon phase, 1 means minimum, can also choose 0
5  fmin=10;fmax=150;fmaxn=60;%post-deconf filter parameters
6  ncc=40;%number of cc lags to examine
7  sd=deconf(s,s,nsmo,stab,phase,'smoothertype',stype);%noiseless
8  [x,str]=maxcorr_phs(r,sd,ncc);%compute cc and phase
9  rb=butterband(r,t,fmin,fmax,4,0);%bandlimit the reflectivity
10 sdb=butterband(sd,t,fmin,fmax,4,0);%bandlimit the decon
11 [xb,strb]=maxcorr_phs(rb,sdb,ncc);%compute cc and phase
12 sdn=deconf(sn,sn,nsmo,stabn,phase,'smoothertype',stype);%noisy
13 [xn,strn]=maxcorr_phs(r,sdn,ncc);%compute cc and phase
14 sdnb=butterband(sdn,t,fmin,fmaxn,4,0);%bandlimit the decon
15 rbn=butterband(r,t,fmin,fmaxn,4,0);%bandlimit the reflectivity
16 [xbn,strnb]=maxcorr_phs(rbn,sdnb,ncc);%compute cc and phase
```

End Code

deconcode/ deconf_test1 .m

a simpler approach of a direct search over integer values of phase. Both give essentially the same answer. So, given two signals to be compared, *maxcorr_phs* determines the absolute maximum crosscorrelation, the lag (in samples) at which this occurs, and the best constant-phase rotation to match one to the other. It should be noted that there is an essential ambiguity between the determination of a time shift (i.e., the lag) and a phase rotation. If the lag quoted here is large (say, greater than half a period), then the phase rotation is probably not very meaningful. Instead, the time shift should be removed before the phase rotation is measured. The function *maxcorr_ephs* addresses this additional complexity

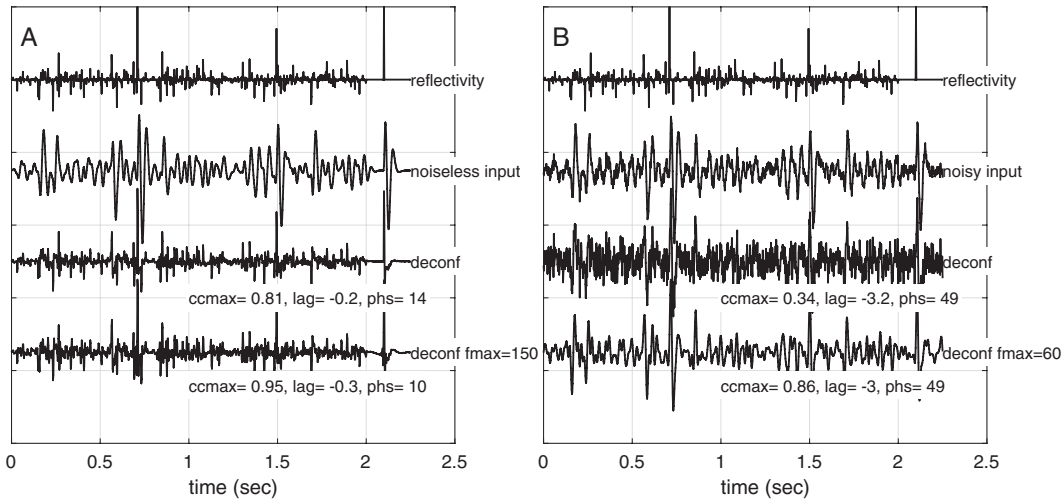


Figure 5.6 The results of Code Snippet 5.3.3 are shown in the time domain. Panel A is for the noise-free case, while panel B is for the noisy case. The reflectivity and input traces are the same as in Figure 5.4 except that an extra isolated spike has been attached to the end of the reflectivity to capture an isolated wavelet after convolution. Annotated beneath each deconvolution result are numerical measures of the comparison with the reflectivity. Shown are the maximum crosscorrelation coefficient, the lag at which it occurs (samples), and the constant-phase rotation required to best match the deconvolved trace to the reflectivity. For the comparison after filtering, both the trace and the reflectivity were band limited.

but is not needed here. The estimated phase is essentially a weighted average of the actual phase of the postdeconvolution embedded wavelet (see Figure 5.5b).

Examination of Figure 5.6 shows that the deconvolution in the noise-free case has done an excellent job. The isolated wavelet at the end of the trace is nicely collapsed and the reflectivity everywhere seems nicely resolved. The trace labeled "deconf" is the broadband deconvolution, and the correlation values annotated below it are in comparison with the broadband reflectivity. The `stab` value used was 10^{-6}, which was mentioned before to correspond to about -60 dB. In Figure 5.7, the spectrum before deconvolution reaches -60 dB at about 150 Hz and the postdeconvolution spectrum shows the whitening trailing off at about this frequency. The trace in Figure 5.6 labeled "deconf fmax=150" compares the trace after a 5–150 band-pass filter (zero phase) with the reflectivity with the same filter, and the maximum correlation value has increased dramatically from 0.81 to 0.95. It is generally the case that maximum correlation values are a strong function of the bandwidth chosen for the correlation. The lag values are in samples, and a negative number means that the trace appears delayed with respect to the reflectivity, as it should because the embedded wavelet is still a minimum-phase imperfect spike. The phase values of 14° and 10° are quite small and very acceptable. Many interpreters believe that the human eye cannot distinguish phase variations of 10° or less, so these values are qualitatively close to zero.

The deconvolution of the noisy trace has been much less successful. In panel B of Figure 5.6, the trace labeled "deconf" is quite unacceptably noisy in spite of the much

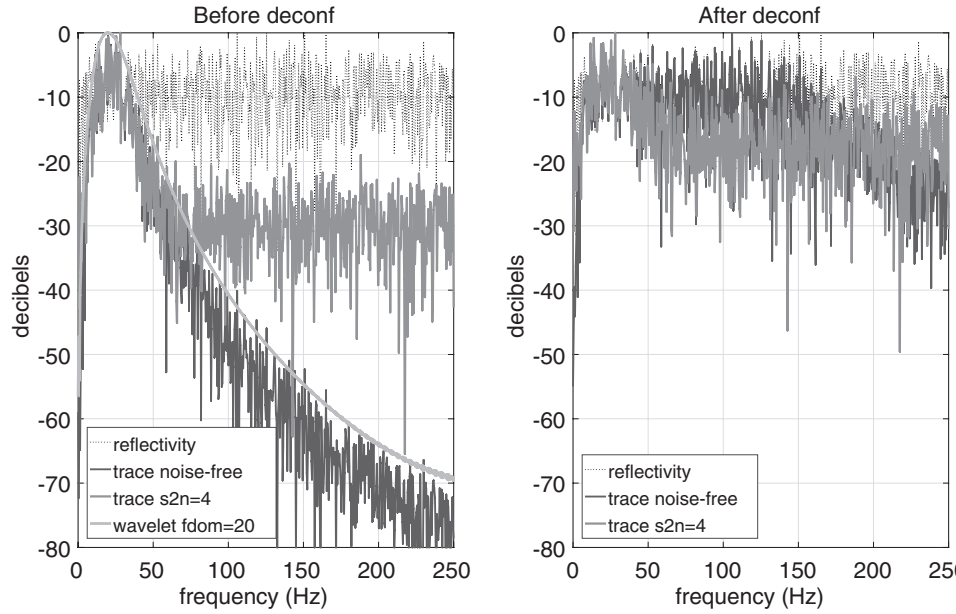


Figure 5.7 The results of Code Snippet 5.3.3 are shown in the frequency domain. On the left are the spectra before deconvolution, while on the right are the spectra after deconvolution. The spectra of the deconvolved and then filtered traces are not shown.

larger `stab` value of 10^{-2}, or about -20 dB. From the spectra of the input trace, we expect this to suppress whitening at around 60 Hz, and this does appear to be the case. It is an unavoidable consequence of the spectral whitening of deconvolution that noise will be enhanced. However, it is a mistake to avoid deconvolution for fear of "blowing up the noise," because the embedded wavelet is then never collapsed. Rather, deconvolution should be followed by one or more processes designed to suppress the noise such as a CMP (common-midpoint) stack or perhaps even a filter. In this case, a 5–60 Hz band-pass filter has resulted in a dramatically improved result, and it is obvious that the filter is far better at noise reduction than trying to adjust the `stab` value (see Exercise 5.3.4). A filter is a good choice for noise suppression if no further deconvolution is planned; otherwise, a subsequent deconvolution will simply remove the filter and restore the noise. Care should always be exercised when judging results based solely on a crosscorrelation measurement. The 0.86 value for the filtered result on the noisy trace falls midway between the two quoted values for the noise-free case, yet both of the noise-free results are far better. It is always important to understand the context of the correlation measurement, especially the bandwidth. The quoted lags and phase rotations are much larger for the noisy case, indicating the wavelet estimation was less successful.

Exercises

5.3.4 Repeat the experiment of Code Snippet 5.3.3 but use 0.000001 for `stab` for both traces. Show that the result for the noisy trace after band-pass filtering is actually improved over the result shown in Figure 5.6.

5.3.5 Repeat an experiment like Code Snippet 5.3.3 but use several different signal-to-noise ratios. Display your results and describe how the signal-to-noise ratio affects the resulting signal band.

5.3.6 Create a stationary convolutional synthetic with noise like that in Code Snippet 5.3.3 and study the quality of the wavelet estimation in *deconf* as a function of the spectral smoother length and type. The second return from *deconf* is the spectrum of the inverse operator, and inverting this will give the estimated wavelet. That is, `wavelet=real(ifft(fftshift(1./specinv)))`.

5.4 Time-Domain Stationary Spiking Deconvolution

The time-domain deconvolution algorithm was developed first and is still the most common approach to be found in industry. Rooted in the digital signal theory of Norbert Wiener, a famous mathematician at MIT during World War II, it was brought to the seismic industry by Enders Robinson (Robinson, 1954, 1967), with further contributions by Peacock and Treitel (1969) and others. The method exists in the *NMES Toolbox* as *deconw*. Every step of the frequency-domain algorithm has its parallel in the time-domain approach. Similar results are easily obtained from either approach, but it is very useful to understand both. The design of the deconvolution operator is the central problem in both algorithms, and this amounts to a statistical method of estimating and inverting the unknown embedded wavelet. However, while inverting the wavelet in the frequency domain is a matter of simple division, in the time domain a different approach is required and this has unique advantages.

5.4.1 Calculating a Wavelet's Inverse in the Time Domain

Let $x(t)$ be the unknown inverse of a wavelet $w(t)$. Then the two signals must satisfy

$$(w \bullet x)\,(t) = \delta(t), \tag{5.20}$$

where $\delta(t)$ is the Dirac delta function. Using the concept of a convolution matrix (Section 3.3.1), a discrete (sampled) equivalent of this equation can be developed. Let $\underline{w}$ be the wavelet of length M as a column vector and $\underline{x}$ be the inverse column vector of length N, where we assume $N < M$. Then, the length of the convolution of $\underline{w}$ and $\underline{x}$ will be

$O = M + N - 1$. Assuming both $\underline{w}$ and $\underline{x}$ to be causal leads to

$$\begin{bmatrix} w_0 & 0 & 0 & \dots & 0 \\ w_1 & w_0 & 0 & \dots & 0 \\ w_2 & w_1 & w_0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{M-1} & w_{M-2} & w_{M-3} & \dots & w_0 \\ 0 & w_{M-1} & w_{M-2} & \dots & w_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \omega_{M-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{5.21}$$

which we recast symbolically as

$$\underline{\underline{W}}\,\underline{x} = \underline{I}, \tag{5.22}$$

where $\underline{\underline{W}}$ is the $O \times N$ convolution matrix of $\underline{w}$ and $\underline{I}$ is the unit vector of length O depicted above. This matrix system of equations has been deliberately constructed with O equations and N unknowns where $O \gg N$, so that the method of *least squares* (e.g., Strang (1986), pp. 34–37) can be employed in its solution. The least-squares method is a powerful approach to finding the "best" solution to a linear system having more equations than unknowns, which is called an *overdetermined* system. The method can be developed intuitively by noting that the matrix $\underline{\underline{W}}$ has no classical inverse, because it is rectangular and only square matrices have inverses. However, we can construct a square system from Eq. (5.22) by multiplying both sides by $\underline{\underline{W}}^{\mathrm{T}}$, the transpose of $\underline{\underline{W}}$, as

$$\underline{\underline{W}}^{\mathrm{T}}\,\underline{\underline{W}}\,\underline{x} = \underline{\underline{W}}^{\mathrm{T}}\,\underline{I}. \tag{5.23}$$

Since $\underline{\underline{W}}$ is $O \times N$, then $\underline{\underline{W}}^{\mathrm{T}}$ is $N \times O$, so that $\underline{\underline{A}} = \underline{\underline{W}}^{\mathrm{T}}\,\underline{\underline{W}}$ is $N \times N$, or square. To further understand $\underline{\underline{A}}$, it helps to explicitly detail $\underline{\underline{W}}^{\mathrm{T}}$:

$$\underline{\underline{W}}^{\mathrm{T}} = \begin{bmatrix} w_0 & w_1 & w_2 & \dots & w_{M-1} & 0 & \dots & 0 \\ 0 & w_0 & w_1 & \dots & w_{M-2} & w_{M-1} & \dots & 0 \\ 0 & 0 & w_0 & \dots & w_{M-3} & w_{M-2} & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & w_0 & w_1 & \dots & w_{M-1} \end{bmatrix}. \tag{5.24}$$

Observe that each row of the matrix in Eq. (5.24) is a column of the matrix in Eq. (5.21). $\underline{\underline{W}}^{\mathrm{T}}$ has constant diagonals, meaning that it too is a convolution matrix, but what vector is it applying? In each column of $\underline{\underline{W}}^{\mathrm{T}}$ is the wavelet, but time-reversed. Therefore, application of $\underline{\underline{W}}^{\mathrm{T}}$ is convolution with the time-reversed wavelet, which is the same thing as crosscorrelation (see Section 2.2). So, Eq. (5.23) is equivalent to $w(-t) \bullet w(t) \bullet x(t) = \underline{I}$. Now, $w(-t) \bullet w(t) = \alpha(t)$ is the autocorrelation of the wavelet, and the matrix $\underline{\underline{A}}$ is the convolution matrix formed from the first N lags of the autocorrelation. Then Eq. (5.23)

becomes

$$\begin{bmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \dots & \alpha_{N-1} \\ \alpha_1 & \alpha_0 & \alpha_1 & \dots & \alpha_{N-2} \\ \alpha_2 & \alpha_1 & \alpha_0 & \dots & \alpha_{N-3} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \alpha_{N-1} & \alpha_{N-2} & \alpha_{N-3} & \dots & \alpha_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} w_0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{5.25}$$

which can be written symbolically as

$$\underline{\underline{A}}\,\underline{x} = w_0 \underline{I}. \tag{5.26}$$

Formally assuming that $\underline{\underline{a}}$ is invertible, the solution to this system of equations is

$$\underline{x}_{ls} = w_0 \underline{\underline{A}}^{-1} \underline{I}. \tag{5.27}$$

In the deconvolution problem, since the wavelet is unknown, it is customary to choose $w_0 = 1$, meaning that the solution is only accurate to within a constant scale factor.

In the language of the signal theory developed by Wiener (e.g., Robinson and Treitel (1967)), the equation system (5.25) is called the *normal equations*, while the system (5.21) is the *design equations*. The latter state the goal of the filter that we are designing, which in this case is to invert a known, causal wavelet. These equations also stipulate that we seek a causal inverse to a causal wavelet, and the alert reader will recognize that we are thus imposing a minimum-phase assumption (see Section 2.4.8).[4] There is no guarantee that our design equations will be exactly met; in fact, it is virtually guaranteed that they will not, because a system with more equations than unknowns usually has no exact solution. However, the normal equations may well have a solution if the square matrix $\underline{\underline{A}}$ proves to be invertible. While it is not shown here, the solution, if it exists, is well known to be a least-squares solution (e.g., Strang (1986)) in the sense that $\left\| \underline{\underline{A}}\,\underline{x}_{ls} - \underline{I} \right\|_2$ is minimized, where $\underline{x}_{ls}$ is the solution to the normal equations and $\|.\|_2$ is the L_2 norm. This means that $\underline{x}_{ls}$ is guaranteed to invert $\underline{w}$ better than any other inverse of the same (or shorter) length.

5.4.2 The Time-Domain Algorithm

The theory in the previous section explains how to calculate the least-squares inverse to a known minimum-phase wavelet. Now the central problem is that we have an unknown wavelet and the implementation of Eq. (5.25) requires that we somehow estimate the autocorrelation of this unknown wavelet. This is strictly analogous to the circumstance in the frequency-domain algorithm where the power spectrum of the wavelet was estimated by smoothing the power spectrum of the trace (recall that the Fourier transform of the autocorrelation is the power spectrum; see Section 2.2). In the present circumstance, the autocorrelation of the wavelet can be estimated from the trace autocorrelation provided that the reflectivity is random. Let $s(t) = w(t) \bullet r(t) + n(t)$ be a noisy trace obeying the

[4] Notice that Eq. (5.25) contains no phase information. Therefore the phase must be coming in via an implicit assumption.

convolutional model. The autocorrelation of this trace is

$$\begin{aligned} s(-t) \bullet s(t) &= [w(-t) \bullet r(-t) + n(-t)] \bullet [w(t) \bullet r(t) + n(t)] \\ &= \underbrace{w(-t) \bullet w(t) \bullet r(-t) \bullet r(t)}_{c_1} + \underbrace{w(-t) \bullet r(-t) \bullet n(t)}_{c_2} \\ &\quad + \underbrace{w(t) \bullet r(t) \bullet n(-t)}_{c_3} + \underbrace{n(-t) \bullet n(t)}_{c_4}. \end{aligned}$$

Considering these terms separately, the first can be written as $c_1 = \alpha_w(t) \bullet \alpha_r(t)$, where α_w and α_r are the autocorrelations of the wavelet and the reflectivity. The terms c_2 and c_3 are both zero because they involve crosscorrelations between the random noise and the random reflectivity. Finally, $c_4 = \alpha_n(t)$, which is the autocorrelation of the noise. So we finally have for the trace autocorrelation

$$\alpha_s(t) = s(-t) \bullet s(t) = \alpha_w(t) \bullet \alpha_r(t) + \alpha_n(t). \tag{5.28}$$

The autocorrelation of an infinitely long random signal is $\delta(t)$, meaning that the signal correlates with itself perfectly at zero lag but for any other lag there is no correlation at all. For a finite-length portion of a random signal, this degenerates slightly to $P\overline{\delta}(t)$, where $\overline{\delta}(t)$ is a very sharply peaked function at $t = 0$ and P is the total power of the signal. So, this equation becomes approximately

$$\alpha_s(t) = P_r \alpha_w(t) \bullet \overline{\delta}(t) + P_n \overline{\delta}(t) \approx P_r \alpha(t) + P_n \delta(t), \tag{5.29}$$

where, in the final form, the approximate deltas have been replaced by true deltas and we have used the fact that convolution of any function with a δ is the identity operation (it does not change the function). So, if the reflectivity and noise are both random, then the expectation is that the autocorrelation of the trace can indeed estimate the autocorrelation of the wavelet. Figure 5.8 shows the signals and their autocorrelations for the synthetic dataset that was used as an example for `deconf`. This reflectivity is computer random, so its autocorrelation is about as close to a true δ as can be expected. The main things to notice in this figure are that the autocorrelations of the noisy and noise-free traces are almost identical and the autocorrelation of the wavelet does resemble that of the traces, but only for the smaller lags.

From Figure 5.8, it follows that the use of the trace autocorrelation values in Eq. (5.25) really only makes sense out to a certain lag, but that critical lag is essentially unknown. It is clearly related to the length of the wavelet, but that too is unknown although we expect it to be much smaller than the trace length. So here is a fundamental uncertainty in the algorithm: the maximum lag of the trace autocorrelation to allow in the operator design must be specified. From Eq. (5.25), it follows that this is the same thing as specifying the temporal length of the deconvolution operator, t_d. This is essentially a windowing action applied to the autocorrelation as a multiplication operator. Simply selecting those lags between 0 and t_d corresponds to multiplying the autocorrelation by a boxcar that extends from $-t_d$ to $+t_d$. In the frequency domain, this corresponds to convolving the power spectrum with the Fourier transform of a boxcar. This should sound familiar, because this is a smoothing operation on the power spectrum and that is exactly what the frequency-domain algorithm does,

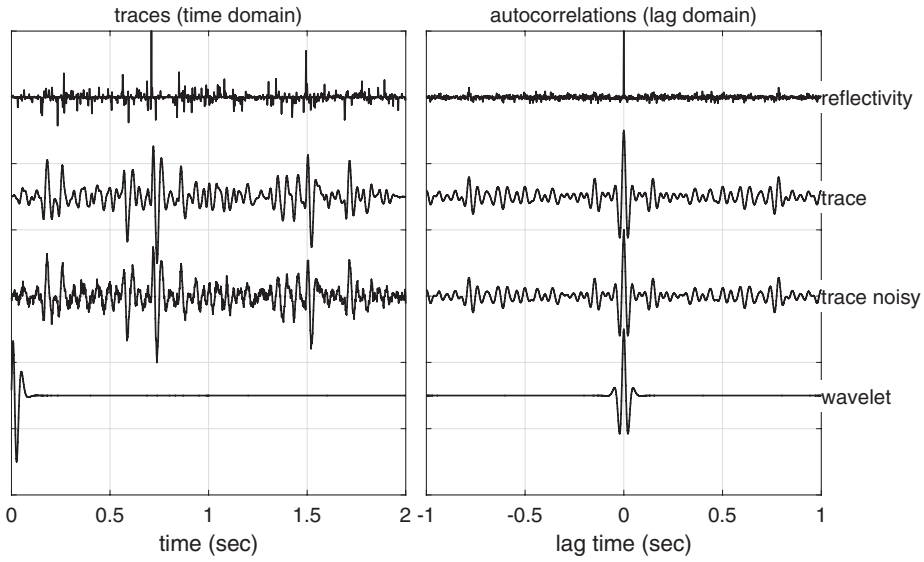


Figure 5.8 The synthetic traces of Figure 5.4 plus the reflectivity and wavelet are shown along with their autocorrelations. All of the autocorrelations have been normalized and were computed with *ccorr*.

Code Snippet 5.4.1 The essential steps of time-domain spiking deconvolution are illustrated, as extracted from *deconw*. The input trace `trin` is deconvolved with an operator designed from the design trace `trdsign`. The number of autocorrelation lags is `n`, and the stability constant is `stab`. Line 1 calls *auto* to compute the one-sided autocorrelation of the design trace. Line 2 adjusts the zero-lag autocorrelation with the stability constant and line 3 sets up the right-hand side of the normal equations (here called `b` instead of $\underline{I}$). Line 4 designs the deconvolution operator by calling *levrec* and line 5 deconvolves `trin` by convolving it with the operator.

```
1     a=auto(trdsign,n,0);% generate the autocorrelation
2     a(1)=a(1)*(1.0 +stab);% stabilize the auto
3     b=[1.0 zeros(1,length(a)-1)];% RHS normal equations
4     x=levrec(a,b);% do the levinson recursion
5     trout=convm(trin,x);% deconvolve trin
```

End Code

deconcode/deconw_guts.m

except that the shape of the smoother is a little different. In fact, applying a boxcar window to the autocorrelation can now be seen to be perhaps risky because the Fourier transform of a boxcar is a sinc function, which has both positive and negative lobes. Smoothing the power spectrum by convolution with a sinc function therefore runs the risk that the smoothed spectrum may have negative values, which is mathematically impossible for a power spectrum. From the time-domain perspective, it may happen that the windowed trace autocorrelation is not the autocorrelation of any possible function, much less the wavelet.

Therefore, other windows may be more sensible, such as a triangle or a Gaussian. Since a triangle is the convolution of a boxcar with itself, it follows that its Fourier transform is a sinc squared, which is nonnegative but does still have possibly problematic zeros. A Gaussian might make more sense, but it is mathematically infinite and therefore will still have to be truncated. Also, it is possible that the true embedded wavelet has no inverse, perhaps because of zeros in its amplitude spectrum, and that might also lead to difficulties in calculation of $\underline{\underline{A}}^{-1}$. With almost any choice of window, there is always the chance that the matrix $\underline{\underline{A}}$ of the normal equations will not have an inverse. Therefore, it is common practice to add a small positive constant to the main diagonal, which, if large enough, will guarantee that an inverse exists. Therefore we modify Eq. (5.25) to

$$\begin{bmatrix} \alpha_0(1+\mu) & \alpha_1 & \alpha_2 & \dots & \alpha_{N-1} \\ \alpha_1 & \alpha_0(1+\mu) & \alpha_1 & \dots & \alpha_{N-2} \\ \alpha_2 & \alpha_1 & \alpha_0(1+\mu) & \dots & \alpha_{N-3} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \alpha_{N-1} & \alpha_{N-2} & \alpha_{N-3} & \dots & \alpha_0(1+\mu) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} w_0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad (5.30)$$

which can be written symbolically as

$$\underline{\underline{A}}_{\mu}\underline{x} = \left[\underline{\underline{A}} + \mu\underline{\underline{I}}\right]\underline{x} = w_0\underline{I}, \quad (5.31)$$

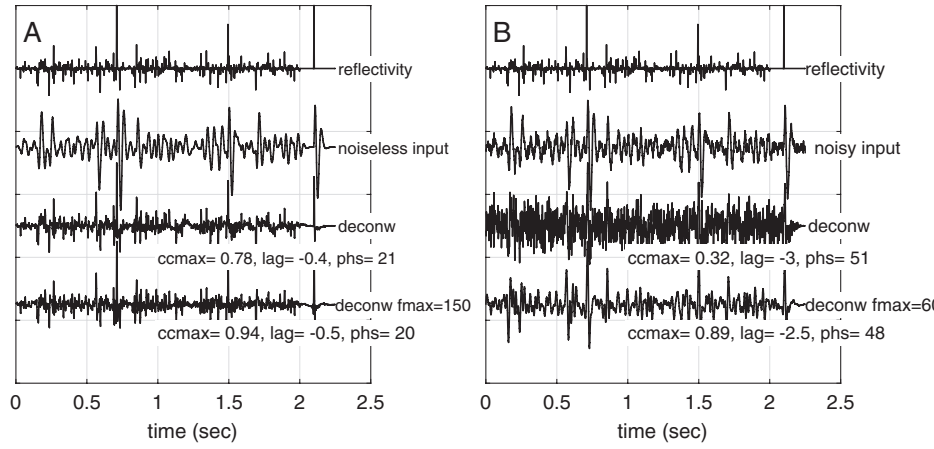


Figure 5.9 The results of Code Snippet 5.4.2 are shown in the time domain. Panel A is for the noise-free case, while panel B is for the noisy case. The reflectivity and input traces are the same as in Figure 5.4 except that an extra isolated spike has been attached to the end of the reflectivity to capture an isolated wavelet after convolution. Annotated beneath each deconvolution result are numerical measures of the comparison with the reflectivity. Shown are the maximum crosscorrelation coefficient, the lag at which it occurs (samples), and the constant-phase rotation required to best match the deconvolved trace to the reflectivity. For the comparison after filtering, both the trace and the reflectivity were band limited. Compare with the frequency-domain result in Figure 5.6.

Code Snippet 5.4.2 This example is meant to parallel that of Code Snippet 5.3.3 and produce similar results, which are shown in Figures 5.9 and 5.10. The `stab` values are different here, for reasons discussed in the text, and were chosen by trial and error to be similar to those used in the frequency-domain example. On line 1, the operator length is chosen as 0.1 s, which is the inverse of the frequency smoother width used in Code Snippet 5.3.3. Lines 6 and 11 accomplish the actual deconvolutions and the parameter `wndw` has been set to use a boxcar window for the autocorrelations. Everything else is quite similar to the frequency-domain test and has already been explained.

```
1  td=0.1;stab=.0001;stabn=.1;%deconw parameters
2  n=round(td/dt);%operator length in samples
3  wndw=1;%window type, 1=boxcar, 2=triangle, 3=gaussian
4  fmin=10;fmax=150;fmaxn=60;%post-deconw filter parameters
5  ncc=40;%number of cc lags to examine
6  sd=deconw(s,s,n,stab,wndw);%noiseless deconw
7  [x,str]=maxcorr_phs(r,sd,ncc);%compute cc and phase
8  rb=butterband(r,t,fmin,fmax,4,0);%bandlimit the reflectivity
9  sdb=butterband(sd,t,fmin,fmax,4,0);%bandlimit the decon
10 [xb,strb]=maxcorr_phs(rb,sdb,ncc);%cc and phase after filter
11 sdn=deconw(sn,sn,n,stabn,wndw);%noisy deconw
12 [xn,strn]=maxcorr_phs(r,sdn,ncc);%compute cc and phase
13 sdnb=butterband(sdn,t,fmin,fmaxn,4,0);%bandlimit the decon
14 rbn=butterband(r,t,fmin,fmaxn,4,0);%bandlimit the reflectivity
15 [xbn,strnb]=maxcorr_phs(rbn,sdnb,ncc);%cc and phase after filter
```

End Code

deconcode/deconw_test1.m

where $\underline{\underline{I}}$ is the $N \times N$ identity matrix. The quantity μ is almost the same thing as the stability constant encountered in the frequency-domain algorithm. Thus the time-domain deconvolution operator follows from the solution of Eq. (5.31) as

$$d(t) = x_{ls} = \underline{\underline{A}}_{\mu}^{-1} \underline{I}, \tag{5.32}$$

where $w_0 = 1$ has been assumed, which means that the overall magnitude of the operator is indeterminate. While Eq. (5.32) gives the formal solution for the deconvolution operator, it is usually computed using a fast method known as the *Levinson recursion* (e.g., Press et al. (1992)). This method solves a Toeplitz symmetric problem like Eq. (5.30) without ever forming the matrix $\underline{\underline{A}}_{\mu}$. Unlike the frequency-domain approach, the operator is naturally minimum phase by virtue of the assumptions made in the structure of the design equations, and no Hilbert transform is required. Code Snippet 5.4.1 illustrates the basic steps of the time-domain algorithm and should be compared with Code Snippet 5.3.1. Superficially, the time-domain method may look simpler, but this is deceptive because there is a considerable amount of complexity hidden in the functions *auto*, *levrec*, and *convm*, while Code Snippet 5.3.1 includes the forward and inverse Fourier transform machinery. A more relevant point of comparison is that the way in which the `stab` value (or μ) is used in the two methods is fundamentally different, and this means that the `stab`

Table 5.1 Comparison of deconvolution algorithms

Step	Frequency domain	Time domain
Isolate design window	Compute power spectrum	Compute autocorrelation
Stabilize	Add $\mu P_{\max}$ to all f	Add $\mu\alpha_0$ to α_0
Estimate amplitude	Smooth power spectrum	Window autocorrelation
Estimate phase and operator	Hilbert transform and invert	Solve normal equations
Apply operator	Frequency-domain multiplication	Time-domain convolution

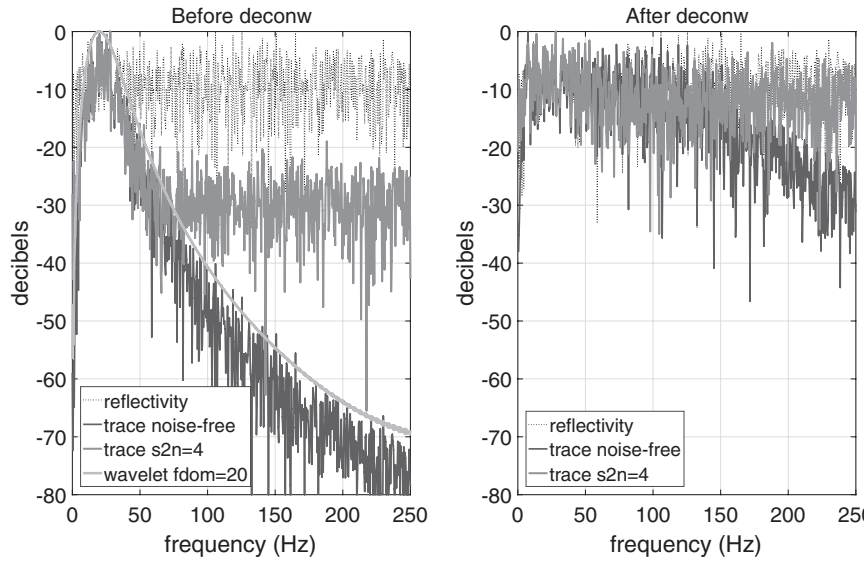


Figure 5.10 The results of Code Snippet 5.4.2 are shown in the frequency domain. On the left are the spectra before deconvolution, while on the right are the spectra after deconvolution. The spectra of the deconvolved and filtered traces are not shown. Compare with the frequency-domain result in Figure 5.7.

values used in *deconw* do not give directly similar results when the same values are used in *deconf*. In the frequency domain, the power spectrum is stabilized by the additive value $\mu P_{\max}$ (Eq. (5.14)), but in the time domain the zero-lag autocorrelation is increased by the additive value $\mu\alpha_0$. While the power spectrum and the autocorrelation are a Fourier transform pair, these additive constants are not the same. The zero-lag autocorrelation is the sum of squares of the trace samples, which, by Parseval's relation (Eq. (2.65)), is essentially the mean power. In order to make the frequency-domain method more similar to the time-domain method, we would need to use the additive power constant μP_{mean} rather than $\mu P_{\max}$, and this exists as an option in *deconf*. The reason to use the maximum power is so that the `stab` constants can be directly related to decibel levels below the maximum power.

Comparing Figure 5.9 with Figure 5.6 suggests strongly that these two methods can be made to give essentially similar results. This conclusion is further bolstered by comparing Figures 5.10 and 5.7. Table 5.1 compares the two algorithms in a step-by-step fashion,

illustrating which steps have approximate equivalence. This equivalence can be demonstrated in a computational experiment, as shown in Code Snippet 5.4.3 and Figure 5.11. To achieve a nearly perfect equivalence, the smoothing operation in the frequency domain must be made equivalent to the autocorrelation windowing in the time domain. Additionally, the stabilization actions must be made as closely equivalent as possible. Since the Fourier transform of a Gaussian is a Gaussian, then a simple choice is to use a Gaussian smoother in frequency and a Gaussian window in time and to make their widths inversely related. This is achieved on lines 1–3, 12, and 13 in the code snippet. A second point is to make the use of the additive stability constant as parallel as possible. As discussed above, this requires using the option in *deconf* to add μP_{mean} to the power spectrum rather than μP_{max}, which is the default. Then the same `stab` values can be used in both *deconf* and *deconw* with very similar results. For deeper understanding, note that $P_{\text{max}} > P_{\text{mean}}$ so

Code Snippet 5.4.3 This code demonstrates the very close equivalence of *deconf* and *deconw* when care is taken to make the parameters as closely parallel as possible. Toward this end, the time-domain operator length and the frequency-domain smoother length are chosen to be inverses of each other (lines 1 and 14). Also, the frequency smoother and the autocorrelation window are both chosen as Gaussians (lines 3 and 16). The `stab` value is set at 0.0001. Then *deconf* is run twice, first with the additive power constant as μP_{max} (line 7) and then with it as μP_{mean} (line 10). *deconw* is run only once (line 17). All deconvolutions and the reflectivity are band limited to the same 10–150 Hz band and the results compared by crosscorrelation (lines 19–22). Figure 5.11 shows the results.

```
1  fsmo=10;stab=.0001;%deconf parameters
2  nsmo=round(fsmo*max(t));%smoother length in samples
3  stype='gaussian';%smoother type
4  phase=1;%decon phase, 1 means minimum, can also choose 0
5  fmin=10;fmax=150;fmaxn=60;%post-deconf filter parameters
6  ncc=40;%number of cc lags to examine
7  sdf=deconf(s,s,nsmo,stab,phase,'smoothertype',stype);%stabopt max
8  sdfb=butterband(sdf,t,fmin,fmax,4,0);%bandlimit the decon
9  %deconf with staboption mean
10 sdf2=deconf(s,s,nsmo,stab,phase,'smoothertype',stype,...
11     'staboption','mean');
12 sdf2b=butterband(sdf2,t,fmin,fmax,4,0);%bandlimit the decon
13 rb=butterband(r,t,fmin,fmax,4,0);%bandlimit the reflectivity
14 td=1/fsmo;%deconw operator length
15 n=round(td/dt);%operator length in samples
16 wndw=3;%window type, 1=boxcar, 2=triangle, 3=gaussian
17 sdw=deconw(s,s,n,stab,wndw);%noiseless deconw
18 sdwb=butterband(sdw,t,fmin,fmax,4,0);%bandlimit the decon
19 [xb,strfb]=maxcorr_phs(rb,sdfb,ncc);%compare sdfb to rb
20 [xb,strf2b]=maxcorr_phs(rb,sdf2b,ncc);%compare sdf2b to rb
21 [xb,strwb]=maxcorr_phs(rb,sdwb,ncc);%compare sdwb to rb
22 [xb,strwf]=maxcorr_phs(sdf2b,sdwb,ncc);%compare sdf2b to sdwb
```

End Code

deconcode/decon_compare.m

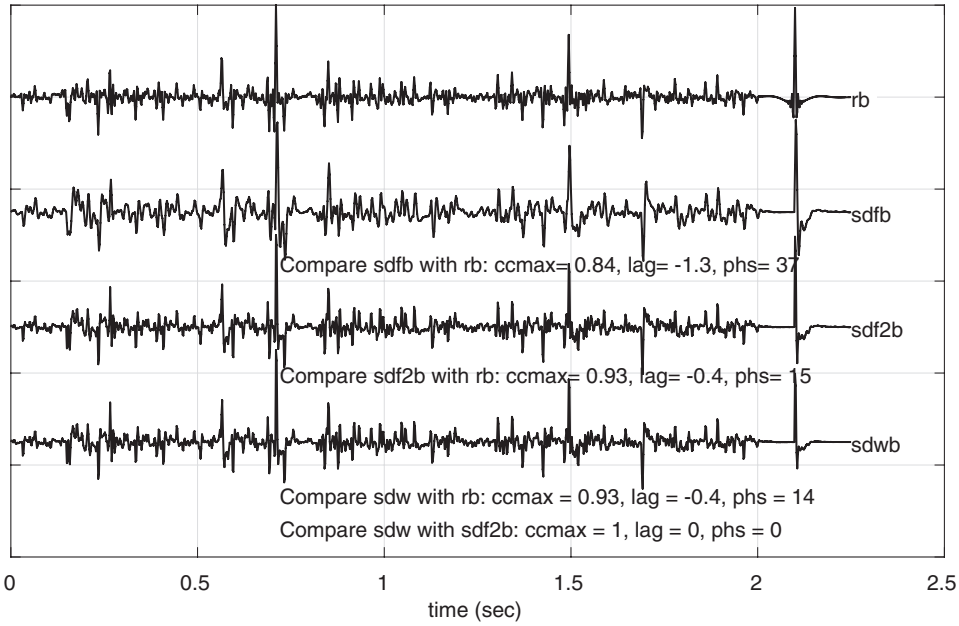


Figure 5.11 Traces, labeled rb, sdfb, sdf2b, and sdwb (from Code Snippet 5.4.3), which are the band-limited reflectivity, the band-limited `deconf` result with $\mu P_{\max}$, the band-limited `deconf` result with μP_{mean}, and the band-limited `deconw` result. Quoted beneath each deconvolution result are the crosscorrelation and phase error measurements. In each case, the input to the deconvolution was the noiseless input trace of Figure 5.9.

that, with the same μ values, the use of $P_{\max}$ does less spectral whitening than P_{mean}. This is apparent in Figure 5.11 both from a visual comparison of the two `deconf` results and from the numerical crosscorrelation and phase measurements. Of course, it is possible to find a smaller `stab` value that will give a comparable result with the $P_{\max}$ option, but this usually requires trial and error. This is because the relationship between the maximum and mean power depends on the detailed shape of the power spectrum. The significant point to be drawn from Figure 5.11 is that the two algorithms are indeed equivalent when their parameters are chosen with care. Indeed, the crosscorrelation of the two deconvolution results has a maximum of 1.0 and a lag of 0, which is the best that can be achieved.

Exercises

5.4.1 Repeat the experiment of Code Snippet 5.4.3 and produce a figure comparing the results in the frequency domain. Discuss your results.

5.4.2 Conduct an experiment with `deconw` using a synthetic convolutional seismogram with additive noise and compare the effect of different operator lengths. Be

sure to determine the signal band of your noisy seismogram and apply a band-pass filter to each result. Describe your results and comment on the importance of the operator length.

5.5 Predictive Deconvolution

It has long been known that the time-domain Wiener deconvolution of the previous section can be generalized through the concept of a prediction operator. In its simplest form, a prediction operator is a convolutional operator that takes N values of a time series and tries to predict the sample value at location $N+1$. Such a filter is called a *prediction filter of unit lag*, where "unit lag" means the prediction is just one sample ahead. The operator length is N and, if causal, it has samples $p_0, p_1, \ldots, p_{N-1}$. By design, these same N operator samples would be expected to predict the next sample in the time series equally well at all locations. The Wiener design equations for a unit-lag, causal prediction operator of length N applied to a causal signal of length M, $s_0, s_1, \ldots, s_{M-1}$, are

$$\begin{bmatrix} s_0 & 0 & 0 & \ldots & 0 \\ s_1 & s_0 & 0 & \ldots & 0 \\ s_2 & s_1 & s_0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{M-1} & s_{M-2} & s_{M-3} & \ldots & s_0 \\ 0 & s_{M-1} & s_{M-2} & \ldots & s_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & s_{M-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{N-1} \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{M-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{5.33}$$

where the convolution matrix, $\underline{\underline{S}}$, has $O = M+N-1$ rows and M columns. The right-hand side of this equation is just the signal advanced by one sample, placing s_1 in the sample 0 location and so on. This is the unit-lag prediction concept. If the lag were, say, 5, then the first sample on the right-hand side would be s_5. Consider the simple case where $N = 2$ and $M = 4$; then the equations are

$$\begin{bmatrix} s_0 & 0 \\ s_1 & s_0 \\ s_2 & s_1 \\ s_3 & s_2 \\ 0 & s_3 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ 0 \\ 0 \end{bmatrix}. \tag{5.34}$$

This is a system of five equations in two unknowns. It has no exact solution, but a least-squares solution can be sought using the techniques of Section 5.4.1. Written explicitly, these five equations are

$$s_0 p_0 = s_1,$$

$$s_1 p_0 + s_0 p_1 = s_2,$$

$$s_2 p_0 + s_1 p_1 = s_3,$$
$$s_3 p_0 + s_2 p_1 = 0,$$
$$s_3 p_1 = 0.$$

Consider the middle three equations, where both filter samples are involved (the other two are special "end" cases). Essentially, we are asking that p_0 and p_1 be multiplicative factors (i.e., weights) such that, when they are combined with any two consecutive samples of $\underline{s}$, the next sample of $\underline{s}$ is the result. In general, this is an impossible task, but, using the least-squares approach, a solution $\underline{p}_{ls}$ can be found such that the prediction is the best possible or that the *prediction error* is minimized. Consequently, a prediction filter becomes a tool whereby an arbitrary signal can be separated into a predictable and an unpredictable part. It will be seen shortly that the unpredictable part of a seismic trace is the reflectivity (it is random, after all), while the predictable part is related to the wavelet. Recalling that the prediction filter is causal by design, this means that it is always trying to use samples earlier in time than a given time to predict the signal at the given time. The fact that the wavelet is an extended causal signal (i.e., not a spike) makes this possible to a large degree. What is not possible is to predict the arrival of a new reflection signal at the given time. Thus the unpredictable part is associated with the reflectivity, although it will require a rescaling.

To proceed further, it helps to use the notation of the z-transform (see Section 3.3.1), because the equations here are greatly simplified and the shift of a signal by j samples is done simply by multiplication by z^j. Using z-transform notation, Eq. (5.33) can be written as

$$s(z)p(z) = z^{-1}\left(s(z) - s_0\right). \tag{5.35}$$

Notice how the right-hand side of Eq. (5.33) is formed by subtracting the first sample of $\underline{s}$ from $\underline{s}$ and then shifting the remaining samples forward (earlier in time) by 1 through the operator z^{-1}. Now, this equation is reexpressed as

$$z^{-1}s(z) - s(z)p(z) = z^{-1}s_0,$$

where the left-hand side is essentially the difference between the predicted values, $s(z)p(z)$, and their actual values, $z^{-1}s(z)$, and is called the *prediction error*. We now multiply through by z and get

$$s(z)(1 - zp(z)) = s(z)\chi(z) = s_0, \tag{5.36}$$

where $\chi(z) = 1 - zp(z)$ is called the *prediction error filter of unit lag*. Equation (5.36) is identical within a scale factor to the z-transform equivalent to Eq. (5.21), the design equations for an inverse filter. For more clarity, we can reexpress Eq. (5.36) as a matrix

equation as

$$\begin{bmatrix} s_0 & 0 & 0 & \dots & 0 \\ s_1 & s_0 & 0 & \dots & 0 \\ s_2 & s_1 & s_0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{M-1} & s_{M-2} & s_{M-3} & \dots & s_0 \\ 0 & s_{M-1} & s_{M-2} & \dots & s_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & s_{M-1} \end{bmatrix} \begin{bmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \\ \vdots \\ \chi_N \end{bmatrix} = \begin{bmatrix} s_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{5.37}$$

where

$$[\chi_0, \chi_1, \chi_2, \chi_3 \dots]^{\mathrm{T}} = [1, -p_0, -p_1, -p_2 \dots]^{\mathrm{T}}. \tag{5.38}$$

Comparing Eqs. (5.21) and (5.37), it is clear that they are equivalent within an overall scale factor: simply divide Eq. (5.37) by s_0 and rename s to w, and the result is Eq. (5.21). Since the estimation of the wavelet inverse is also uncertain by a scale factor, the conclusion is that there is no practical difference between the deconvolution inverse operator and a prediction error filter of unit lag.

An explicit application of the unit-lag prediction error filter to a z-domain signal $s(z)$ is

$$s_d(z) = s(z)\chi(z) = s(z)\,(1 - zp(z)) = s(z) - s_p(z), \tag{5.39}$$

where $s_p(z) = zp(z)s(z)$ is the *predictable part* of $s(z)$. So, a deconvolution can be achieved by computing the prediction filter and using it to form the predictable part of the trace. Then delaying and subtracting this from the original signal gives the unpredictable part, which we identify as the reflectivity. So, rather than computing the inverse of the unknown wavelet, focus is turned to computing a prediction filter directly from the trace. From Eq. (5.33), the normal equations can be formed by multiplying from the left by the transpose of the convolution matrix $\underline{\underline{S}}$. For exactly the same reasons as discussed in the previous section, we also add a small increment to the zero-lag value, $\mu\alpha_0$, with $0 \leq \mu < 1$, to get

$$\begin{bmatrix} \alpha_0(1+\mu) & \alpha_1 & \alpha_2 & \dots & \alpha_{N-1} \\ \alpha_1 & \alpha_0(1+\mu) & \alpha_1 & \dots & \alpha_{N-2} \\ \alpha_2 & \alpha_1 & \alpha_0(1+\mu) & \dots & \alpha_{N-3} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \alpha_{N-1} & \alpha_{N-2} & \alpha_{N-3} & \dots & \alpha_0(1+\mu) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_N \end{bmatrix}. \tag{5.40}$$

Comparing this with the normal equations for the deconvolution operator, Eq. (5.30), shows that only the right-hand side is different. So, the solution can proceed in almost the same way as before using the Levinson recursion (see Code Snippet 5.4.1), and the essential steps are shown in Code Snippet 5.5.1. Essentially, all that needs to be done is to compute the prediction operator from the design trace, apply this operator to the trace being deconvolved, delay the prediction, and subtract it from the original trace. In this example, the design trace is assumed to be the same as the trace being deconvolved, while usually in practice it will be some selected portion of the trace. The solution of the normal equations

Code Snippet 5.5.1 The essential steps of predictive deconvolution are illustrated. The first line establishes `nlag` and `nop`, which are the prediction lag and the operator length in samples. The second line defines the stability constant `stab`. Lines 3–7 compute the prediction filter `prfilt`, line 8 applies it to the trace `s`, and line 9 delays the predictable part by `nlag` and subtracts it from the input. The prediction filter is designed by computing the necessary autocorrelation lags, `a`, of the trace, `s`, and constructing the right-hand side of the normal equations, `b`. These are then passed to *`levrec`* for solution by the Levinson recursion. The delay and subtraction of the predictable part on line 9 assumes that the trace `s` is a column vector. The results are illustrated in Figure 5.12. This is extracted from *`predict`* and *`deconpr`*.

```
1   nlag=1;nop=80;%prediction lag and operator length (in samples)
2   stab=.00001;%stability constant
3   a=auto(s,nlag+nop,0);%one-sided auto
4   a(1)=a(1)*(1.0 +stab);% stabilize the auto
5   a=a/a(1);%normalize
6   b=a(nlag+1:nlag+nop);% RHS of pred filt normal equations
7   prfilt=levrec(a(1:nop),b);% do the levinson recursion
8   spre=conv(s,prfilt);%the predictable part of s
9   sun=s-[zeros(nlag,1); spre(1:length(s)-nlag)];%unpredictable
```

End Code

deconcode/ prefilt .m

can be done in a variety of ways but is done here with the Levinson recursion, as found in *`levrec`*. The details of this algorithm (the Levinson recursion) are well documented and unimportant in the present context. Given the one-sided autocorrelation and the right-hand side, Eq. (5.40) is completely defined and *`levrec`* finds the solution. The reason for the delay of the predicted part before the subtraction is that the first sample of the prediction is always placed at the beginning of the predicted time series even though it is a prediction of the sample at location `nlag` of the input time series. Therefore, line 9 in Code Snippet 5.5.1 precedes the predicted part with `nlag` zeros, thus delaying it properly before subtraction.

The results of the computation in Code Snippet 5.5.1, using the same noiseless input trace as in previous examples, are shown in Figure 5.12. At first glance, the predicted part of the trace seems almost identical to the trace, although slight differences can be discerned with careful examination. The difference between the trace and its predicted part, the unpredictable (or random) part, is much weaker in amplitude than the predicted part. However, when scaled to the same maximum value as the reflectivity, it is seen to be an excellent reflectivity estimate. This scaling ambiguity is present in all deconvolution methods and essentially arises because if $w \bullet s_d$ is a good trace model, then $(w/a) \bullet (as_d)$, where a is any nonzero constant, is an equally good model.

The original purpose of gapped deconvolution was to deal with simple multiples such as the water-bottom multiple encountered in marine data. Multiples that are caused by a reflector above the deconvolution design window can be considered as a part of an extended wavelet and potentially deconvolved, or collapsed, into a single approximate

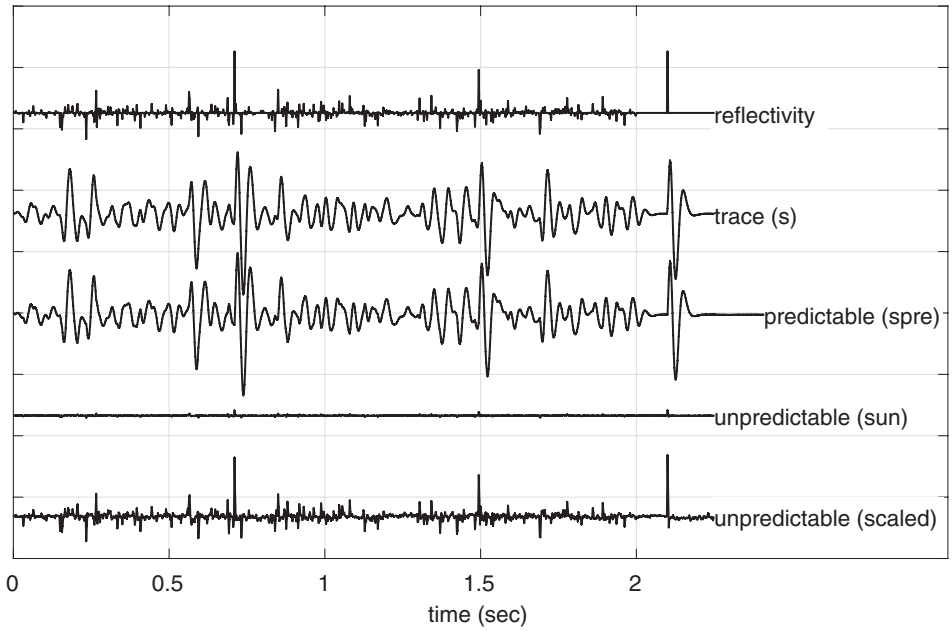


Figure 5.12 The results from Code Snippet 5.5.1 are shown for the noise-free input trace of Figure 5.9 used in previous examples. The input trace, `s`, the predictable part, `spre`, and the unpredictable part, `sun`, are all illustrated. The unpredictable part is shown exactly as computed and after rescaling. The unpredictable part is tiny compared with the trace but after rescaling is seen to be an excellent estimate of the reflectivity.

spike. If the multiple-generating horizon falls within the design window, then it will appear as an extended wavelet for only a subset of the reflectors, meaning that the problem violates the stationarity assumption of the convolutional model. However, the water bottom occurs at the top of the marine stratigraphy, and so multiples associated with the water layer will affect all reflectors and can be considered as forming an extended wavelet. Backus (1959) analyzed the problem of water-layer reverberations and developed a simple deterministic inverse filter that was effective in many cases. However, the Backus filter required explicit knowledge of the water-bottom reflection coefficient and the water-layer traveltime. Predictive deconvolution offered another possibility that did not require this information. Figure 5.13 illustrates the forward modeling of a synthetic trace, including the reverberations as described by Backus's theory. The extended wavelet was created by convolving the source wavelet with the response from *waterbtm*, which requires as input the two-way traveltime through the water layer and the reflection coefficient of the water bottom. These values were 0.2 s and 0.5, respectively. As can be seen, the extended wavelet consists of the original wavelet followed by copies of itself of alternating polarity and decreasing amplitude. The autocorrelations of the trace with and without multiples show considerable differences at lags of about the water traveltime (the dashed lines in the figure). Since the extended wavelet is a series of multiples, there are also autocorrelation anomalies at other integer multiples of this time.

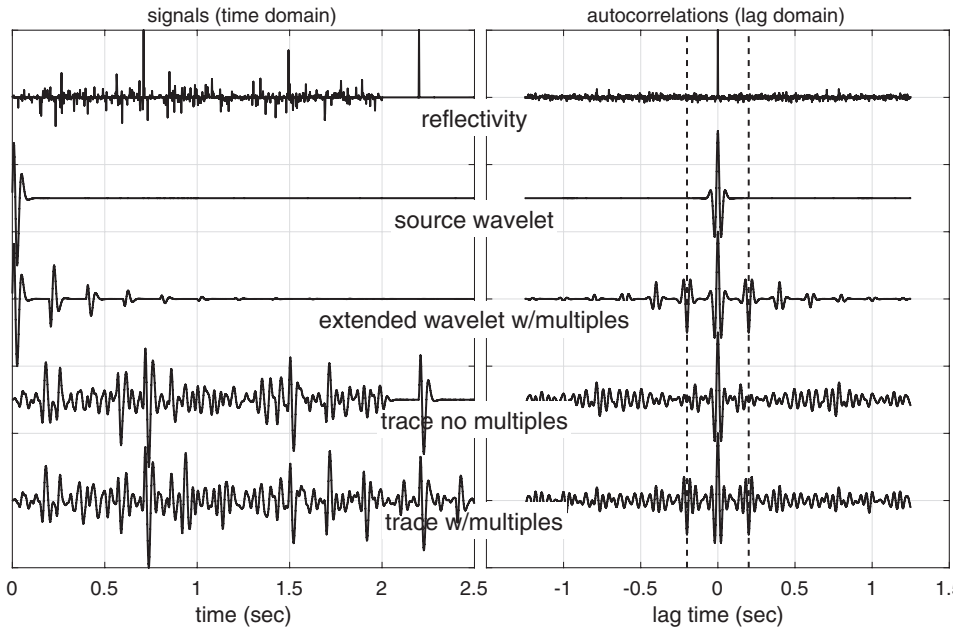


Figure 5.13 Left, the signals, and right, their autocorrelations. Shown are the reflectivity, a minimum-phase source wavelet, the extended wavelet showing the water-layer reverberations, a convolutional trace with only the source wavelet, and a convolutional trace with the extended wavelet. The water-layer reverberations are described by Backus (1959) and the two-way water traveltime was 0.2 s, while the water-bottom reflection coefficient was 0.5. This figure was created by `deconcode/watermult.m`.

The deconvolution of a trace with the extended wavelet, in theory, can be done with spiking deconvolution but requires a very long operator. The operator length is determined by the number of autocorrelation lags used in the design process, and these must include the altered part of the autocorrelation near the water-layer traveltime. In the case of the data of Figure 5.13, the autocorrelation lags out to around 0.3 s are required. An operator length of 0.3 s requires a significantly longer computation time, and back in the 1970s this was problematic. The solution was to use a gapped prediction operator, with the gap (i.e., lag) chosen to be just less than the water traveltime and the operator length perhaps around the usual 0.1 s. This would collapse the multiples and would be followed by a spiking deconvolution with a relatively short operator to collapse the final wavelet. Code Snippet 5.5.2 accomplishes three different deconvolution strategies (using four deconvolutions) on the synthetic trace with multiples. These are spiking deconvolution with a short (0.1 s) operator, spiking deconvolution with a long (0.3 s) operator, and gapped predictive deconvolution (gap 0.19 s, operator 0.1 s) followed by a spiking deconvolution with a short operator. For each case, including the input trace, numerical comparison is made with the band-limited reflectivity after band limiting the deconvolutions. In assessing these results, notice that there are at least two clearly isolated multiples on the input trace before and after the isolated reflectivity spike at 2.2 s. The short-operator deconvolution has failed to

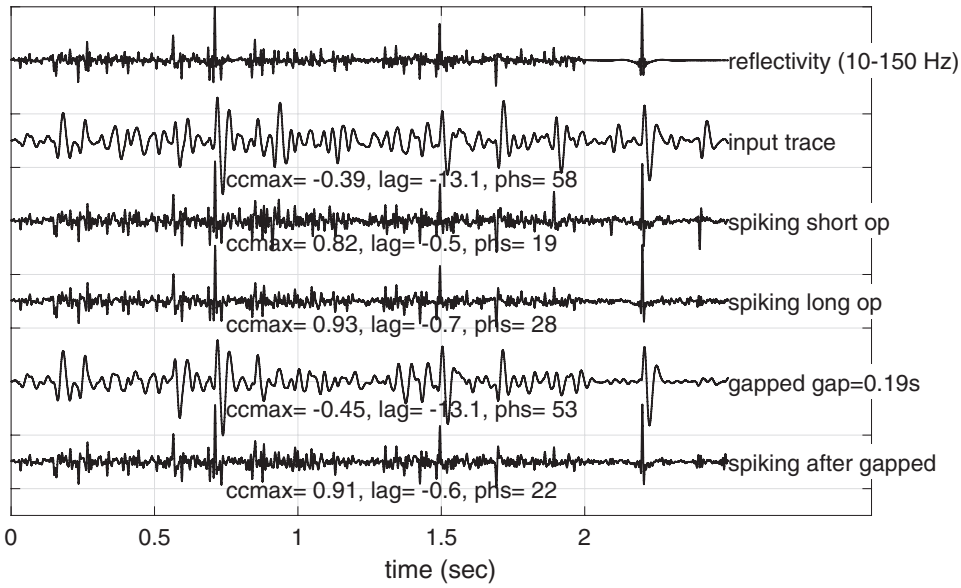


Figure 5.14 The results from Code Snippet 5.5.2. At the top is the band-limited reflectivity, with which everything is compared. The remaining traces have correlation and phase measurements annotated beneath them, from comparison with the band-limited reflectivity. The measurements are (i) the maximum (in absolute value) correlation coefficient, (ii) the lag in samples at which the maximum occurs, and (iii) the apparent constant-phase rotation. Computed by *maxcorr_ephs*, the phase rotation is estimated after shifting the trace by the indicated lag.

collapse these into the source wavelet, and hence they result in false reflectivity spikes. There are presumably other errors elsewhere in the result that are harder to see. Still, the maximum correlation, lag, and phase rotation are all quite good (which indicates that these numbers never tell the entire story). In contrast, the long operator has collapsed at least the first-order multiple into the primary, and the obvious false reflection spikes are absent. The result is a higher correlation but also, strangely, a higher phase rotation. The gapped deconvolution has also attenuated the obvious multiples but there has been almost no whitening. This is characteristic of gapped deconvolution and is why the result should be followed by a spiking deconvolution. This result is still minimum phase, because the extended wavelet is minimum phase and so is the prediction operator. So, a second spiking deconvolution can be legitimately applied to the gapped result and gives a very good reflectivity estimate, essentially as good as the long operator. Today, there seems to be little reason to favor the gapped → spiking approach over the long-operator approach, as both give comparable results and the second requires only one operation. Moreover, if frequency-domain deconvolution is used, the computation time is almost independent of operator length.

When first introduced, gapped predictive deconvolution seemed to offer a great advantage for multiple attenuation; however, it was soon realized that at any significant source–receiver offset the multiples lose their perfectly regular spacing. Instead, their spacing becomes time dependent and so the problem becomes nonstationary (i.e., the extended

Code Snippet 5.5.2 This code accomplishes three different deconvolution strategies for the water-layer multiple synthetic, `sm`, shown in Figure 5.13. Lines 1–9 establish parameters for the deconvolutions, the subsequent filter, and the crosscorrelation analysis. Lines 10–13 accomplish four different deconvolutions: spiking with the short operator, spiking with the long operator, gapped deconvolution with the short operator, and spiking deconvolution with the short operator on the result from the gapped deconvolution. Lines 14–17 apply the same 10–150 band-pass filter to the reflectivity and the deconvolutions, and the remaining lines perform crosscorrelation and phase analysis, comparing each deconvolution with the band-limited reflectivity. The gapped deconvolution was not filtered prior to being input to the spiking deconvolution. The results are shown in Figure 5.14.

```
1   tgap=0.19;%prediction gap; (seconds)
2   td=0.1;%short decon operator (seconds)
3   td2=.3;%long decon operator (seconds)
4   stab=0.0001;%stability constant
5   n=round(td/dt);%short operator length in samples
6   n2=round(td2/dt);%long operator length in samples
7   ngap=round(tgap/dt);%prediction gap in samples
8   fmin=10;fmax=150;%post-decon filter parameters (Hz)
9   ncc=40;%number of cc lags to examine
10  smd=deconpr(sm,sm,n,1,stab);%spiking deconpr short operator
11  smd2=deconpr(sm,sm,n2,1,stab);%spiking deconpr long operator
12  smdg=deconpr(sm,sm,n,ngap,stab);%gapped deconpr
13  smdgd=deconpr(smdg,smdg,n,1,stab);%spiking deconpr after gapped
14  rb=butterband(r,t,fmin,fmax,4,0);%bandlimit the reflectivity
15  smdb=butterband(smd,t,fmin,fmax,4,0);%bandlimit spiking short op
16  smd2b=butterband(smd2,t,fmin,fmax,4,0);%bandlimit spiking long op
17  smdgdb=butterband(smdgd,t,fmin,fmax,4,0);%bandlimit spiking+gapped
18  [x,strm]=maxcorr_phs(rb,sm,ncc);%compute cc and phase
19  [x,strmdb]=maxcorr_phs(rb,smdb,ncc);%compute cc and phase
20  [x,strmd2b]=maxcorr_phs(rb,smd2b,ncc);%compute cc and phase
21  [x,strmdg]=maxcorr_phs(rb,smdg,ncc);%compute cc and phase
22  [x,strmdgdb]=maxcorr_phs(rb,smdgdb,ncc);%compute cc and phase
```

End Code

deconcode/deconpr_water.m

wavelet becomes time-variant). This causes gapped predictive deconvolution to degrade, and other approaches are now often preferred for multiple attenuation.

5.5.1 Using Gapped Deconvolution to Avoid Noisy-Data Problems

It should be obvious by now that the application of deconvolution to any real signal will always require consideration of noise levels. It helps to think of a deconvolution operator as having a spectral-whitening action and a phase action. Since the assumption about the reflectivity is that its spectrum is white, then any estimate of reflectivity should have a whitening action. However, the presence of background noise in real data together with the decay of higher signal frequencies means that there will always be some high frequency

above which noise dominates. Attempting to whiten the spectrum past this point is asking for trouble and usually produces objectionable results. In the examples in this book, this has been mostly addressed with a postdeconvolution band-pass filter that reduces the bandwidth of the deconvolution to what is the anticipated *signal band*.

With synthetic data,it is easy to determine f_{max}, the highest signal-dominated frequency, but with real data, this is much more difficult. Most practical strategies involve postponing the decision until after stack. This is because a CMP stack (for unmigrated data) or a CIG[5] stack (for prestack migrated data) is a very effective suppressor of random noise. So, a preliminary CMP stack can be examined for the highest frequency that shows spatial coherence, and this can be used for f_{max}. It is also a good idea to avoid prestack filters in the main processing flow because a good practice is to deconvolve the data again after stack. This is because the stacking process will cause a dewhitening and a consequent loss of resolution. A deconvolution after stack, followed by a band-pass filter, will generally improve bandwidth and resolution in the final product. This difficulty in dealing with noise while also deconvolving the data has led to an often problematic practice of using gapped deconvolution to reduce spectral whitening and therefore reduce the blowup of noise. Examining Figure 5.14, it is easy to understand where this idea had its origin. Although there is no noise in this simulation, the conclusion that gapped deconvolution has not whitened the data like spiking deconvolution is unavoidable. Therefore, the question becomes not "What f_{max} is best?" but rather "What gap size is best?" Advocates of this approach will claim that the gap size is much easier to choose than f_{max} and a single value can suffice for a wide variety of data. It turns out that when the prediction gap is chosen to control whitening, it is usually a much smaller value than that when chosen for something like the water-layer multiples. In fact, a common choice is something like 0.01 s, which is less than the likely wavelet size. Critics of this approach (the authors are in this camp) suggest that short-gapped deconvolution may do serious harm to the embedded wavelet that will lead to later difficulties in tying data to wells.

Figure 5.15 shows an experiment designed to compare deconvolution of noisy data with either spiking deconvolution followed by band-pass filtering, or gapped predictive deconvolution. The input signals were the same synthetic traces as before (shown in Figure 5.9). The two bottom traces on the left-hand side of Figure 5.15 are the unfiltered results from the noisy input after spiking deconvolution and after gapped deconvolution with a 0.01 s gap. Here it is easy to understand the preference for the gapped deconvolution because that result is far more recognizable, while the spiking deconvolution has produced a noisy mess. Examining the two traces just above these for the noiseless input suggests a preference for spiking deconvolution because it has a much higher maximum correlation and a smaller residual phase. Application of the 10–150 Hz band-pass filter clarifies things. Now the two results on the noisy trace are very similar, although the spiking-deconvolution result shows significantly smaller phase. On the noiseless input, spiking deconvolution is still preferred. Clearly, these results could be changed by choosing a different prediction lag: a smaller value would move the result toward that of spiking deconvolution, while a larger

[5] Common-image gather.

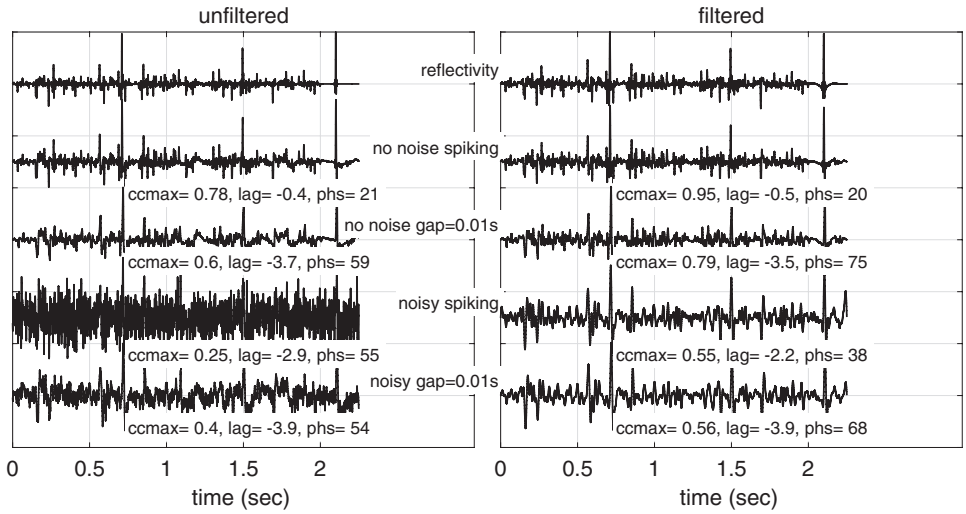


Figure 5.15 The results of an experiment comparing gapped deconvolution to control noise versus spiking deconvolution and postdeconvolution filtering. The input signals were the noiseless input and the noisy input of Figure 5.9. At left are unfiltered traces, while at right are the traces with the same 10–150 Hz band-pass filter. In this case, the prediction gap is 0.01 s, which is a common choice. Down the center of the figure are trace names. Below each trace are three statistics comparing the trace with the reflectivity (top of column): the maximum crosscorrelation, the lag of the maximum (in samples), and the best constant-phase rotation. Compare with Figure 5.16. This experiment was created by the script `deconcode/gapped_decon_all`.

value would result in less noise amplification. In fact, based purely on subjective noise assessment, many might argue that the 0.01 s gap used here is too small because the unfiltered noisy trace looks perhaps too noisy. Figure 5.16 shows the results when the gap is increased to 0.026 s, and the noisy input after gapped deconvolution now looks very clean. Many might prefer this. However, after filtering, the result shows a negative maximum correlation, a large lag, and a large residual phase, all indicating a poor reflectivity estimate. The filtered result after spiking deconvolution correlates much better with the reflectivity. Even on the noiseless input, the gapped deconvolution now shows a poor correlation and a large phase error.

More insight into this issue is found in the results shown in Figure 5.17. Here the prediction operators that produced Figures 5.15 and 5.16 are applied to a single isolated wavelet to see their effect more clearly. The operators were those designed from the noiseless input, so they are of high quality. The spiking deconvolution was produced by a prediction operator of unit lag (meaning a one-sample lag), and its results appear twice in Figure 5.17. Applied to the isolated wavelet, this operator predicts almost everything, except there is a small prediction error at the first sample, shown as the unpredicted part. When scaled up, this becomes the reflectivity estimate, which should just be a single spike in this case. Consider the case now when the gap is five samples. Now the prediction begins at sample 6, the first five samples being unpredicted. Then the unpredicted part is the difference between this prediction and the input and essentially amounts to the original wavelet truncated after

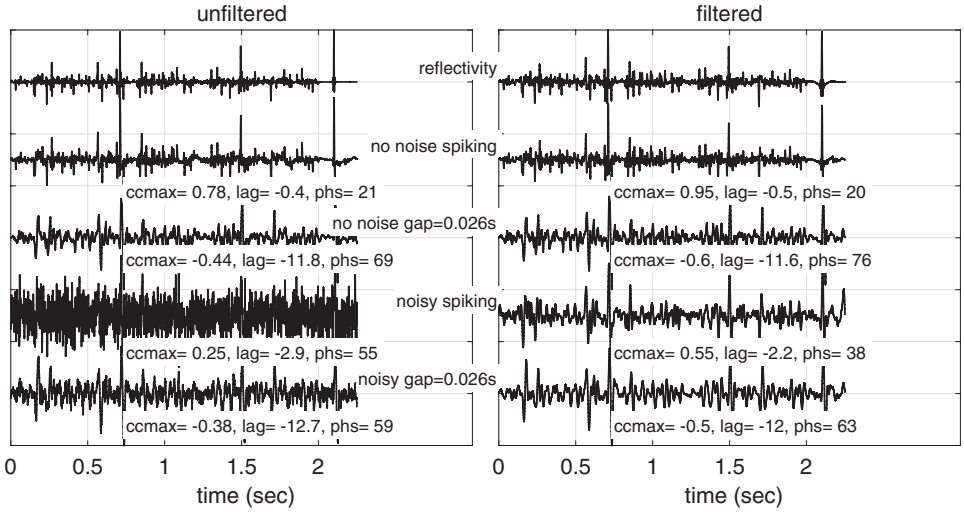


Figure 5.16 The results of an experiment comparing gapped deconvolution to control noise versus spiking deconvolution and postdeconvolution filtering. The input signals were the noiseless input and the noisy input of Figure 5.9. At left are unfiltered traces, while at right are the traces with the same 10–150 Hz band-pass filter. In this case, the prediction gap is 0.026 s, which was chosen to illustrate possible problems. Down the center of the figure are trace names. Below each trace are three statistics comparing the trace with the reflectivity (top of column): maximum crosscorrelation, lag of the maximum (in samples), and the best constant-phase rotation. Compare with Figure 5.15. This experiment was created by the script `deconcode/gapped_decon_all`.

five samples. Using a longer prediction operator would make this truncation sharper. This unpredicted wavelet is then the embedded wavelet remaining in the trace after the gapped deconvolution with a gap of 5. Whether this wavelet is minimum phase seems doubtful. The prediction operator is constructed as minimum phase, but it is the subtraction that is the concern. The convolution of two minimum-phase signals is minimum phase, but what about their subtraction? The answer is a definite "maybe" and it depends on the circumstances. A clear, but almost trivial, example is the subtraction of two impulses separated by some lag. The impulses themselves are minimum phase, but what about their subtraction? If the first impulse is larger (in absolute value), then the result is minimum phase, otherwise it is not. Or consider the case in the previous section of the prediction of a waterlayer multiple where the source is minimum phase. If the multiple is clearly separated in time from the primary and the prediction lag is chosen to be greater than the length of the primary, then all the subtraction does is kill the multiple, so the result is minimum phase. But if they are not clearly separated, then it becomes less clear. In the case examined here, it is not clear whether the unpredicted wavelets of Figure 5.17 can be considered to be minimum phase or not. This is an important question if a second, perhaps poststack, deconvolution is planned. Leaving the gapped deconvolution results as is without a further deconvolution may satisfy a subjective bias against a noisy result but it certainly does not give a good reflectivity estimate. If these wavelets are minimum phase, then a second deconvolution may improve the gapped results. In fact, in a practical setting, the best test

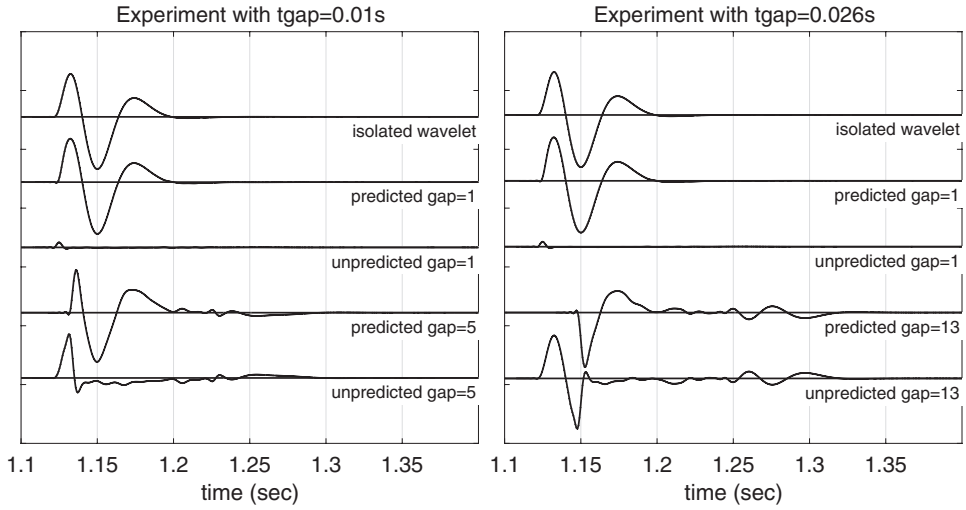


Figure 5.17 This illustrates the effect of the prediction operators of Figures 5.15 and 5.16 when applied to a single isolated wavelet. On the left is the case for the short 0.01 s gap, while on the right is that for the longer 0.026 s gap. Also shown in each panel are the isolated wavelet and the case when the gap is a single sample. Trace labels quote the prediction gap in samples (the time sample size is 0.002 s), so the three gaps are 1, 5, and 13 samples. Dotted lines indicate the zero-amplitude level. In each case a predicted and an unpredicted wavelet are shown. The predicted wavelet is found by applying the prediction operator computed for the noiseless data of either Figure 5.15 or Figure 5.16 to the input wavelet. The unpredicted part then follows from the subtraction of the prediction from the input. This experiment was created by the script `deconcode/gapped_decon_all`.

of whether or not a wavelet is minimum phase is to apply a spiking deconvolution to it. If the result is a good approximation to a band-limited spike, then the input was, practically speaking, minimum phase. Otherwise, it was not. Figure 5.18 shows the results of such a test applied to the three unique deconvolution wavelets of Figure 5.17. These three wavelets are those labeled "unpredicted" and, as mentioned previously, are the embedded wavelets after deconvolution in Figures 5.15 and 5.16. For this test, the three wavelets were all normalized to a maximum absolute value of 1 and then passed through Wiener spiking deconvolution (*deconw*). After deconvolution, they were all band-pass filtered (zero phase) to the same 10–150 Hz band and renormalized. The results show clearly that the wavelets after gapped deconvolution are not very close to minimum phase, although the smaller gap gives a better result.

So, it seems clear that the practice of using gapped deconvolution to control noise amplification is not optimal. Much better is to use spiking deconvolution followed by some sort of noise-reduction process and, perhaps after stack, a second deconvolution. As was stressed here, a second deconvolution is only advised if it can be reasonably surmised that the embedded wavelet is still minimum phase. If the first deconvolution was gapped (with a gap larger than 1), then this is in serious doubt. Certainly, it is reasonable to conclude that a very small gap of perhaps two to three samples could lead to a reasonable result, but why do this when there are much better choices? Gapped predictive deconvolution

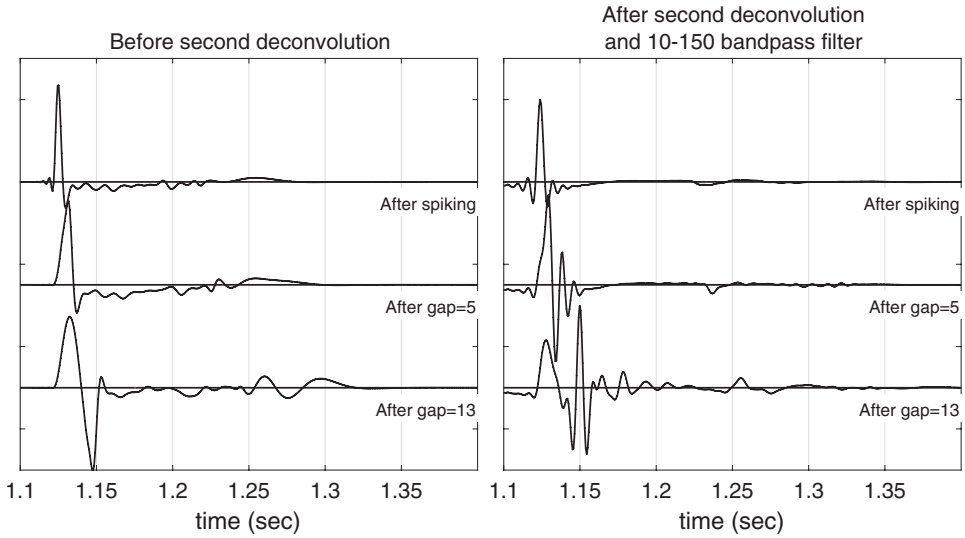


Figure 5.18 The three postdeconvolution wavelets of Figure 5.17 are shown before and after a second spiking deconvolution, which tests if they are minimum phase. Only a minimum-phase wavelet will produce a band-limited spike after this test. The wavelets on the left are the unpredicted wavelets (normalized to 1) of Figure 5.17, while those on the right are the wavelets after a second Wiener spiking deconvolution. In each case the deconvolution parameters were an operator length of 0.1 s and a stability constant of 0.0001. All were band-pass filtered to the same 10–150 Hz band. Both of the gapped deconvolution results fail the test. This experiment was created by the script `deconcode/gapped_decon_all`.

should be used to address long-delay multiples and with a choice of gap that is longer than the presumed source wavelet. This should always be followed by a second spiking deconvolution.

5.6 Nonstationary Deconvolution

The deconvolution methods described so far have been built on an explicitly stationary trace model and result in an explicitly stationary deconvolution process. Yet, as has been discussed in Section 4.7.3, all seismic data is fundamentally nonstationary. There are many possible meanings of the term "nonstationary," so, to be clear, in the context of deconvolution we mean that the seismic wavelet is evolving as it propagates and therefore is a function of time. This is distinct from the fact that any real source emits a signal that is a time series. A stationary trace model posits that whatever waveform the source emits propagates without change and so an identical waveform is incident upon each reflector. This assumption then leads to the convolutional model, whereby a trace is formed as a wavelet convolved with a reflectivity. Mathematicians call convolution a *translation-invariant* process because it simulates propagation (spatial or temporal translation) without

variation. However, this is not what happens in the real Earth, where it is observed that the source waveform does evolve in a very specific way. Owing to anelastic loss, or attenuation (Kjartansson, 1979), the wavelet progressively loses higher frequencies and undergoes the consequent phase changes required of a minimum-phase process. This wavelet evolution is illustrated in Figure 4.17a. The wavelet also evolves because of short-path multiples, as was first described by O'Doherty and Anstey (1971). The first-order effect of short-path multiples is to act as an apparent Q that augments the intrinsic Q of rocks. So, to first order, both processes can be considered through a nonstationary trace model that allows for a time-dependent effective Q. In Section 4.7.4, the concept of a Q matrix was introduced, which allows a progressively decaying Q wavelet to be applied to a reflectivity series, thus giving a nonstationary convolutional trace model (see Figures 4.19 and 4.20). Here *nonstationary convolution* is meant as a generalization that preserves the linear superposition property, but not the translation invariance, of normal, or stationary, convolution.

This discrepancy between the algorithm and the real world is not uncommon, especially in a practical field like seismic exploration. Nor are practicing scientists unaware of the mismatch. Even in 2017, the methods employed in seismic imaging are based on a greatly simplified physics model in order to make them computationally feasible. The stationary deconvolution algorithms are actually a brilliant blend of insight and practicality that is also computationally feasible. These methods were developed before the microchip revolution, when digital computers were extremely limited in their abilities while also being very expensive. Stationary deconvolution of every trace in a seismic dataset was a very challenging computational task, remaining significant even today, and strategies have been developed to allow the method to partially cope with attenuation. As discussed at the beginning of this chapter, amplitude recovery (or gain) was used to adjust the trace amplitudes to a subjective assessment of what was expected in preparation for deconvolution. In doing so, it was always found that the amplitude correction required was greater than that expected for simple wavefront spreading, with the extra being likely due to attenuation. A more important strategy is the idea of using a *design trace* to estimate the deconvolution operator, which is then applied to the entire trace of interest. Usually, the design trace is a segment of the trace being deconvolved that encompasses the zone of interest, meaning a time window that spans the exploration target but is much smaller than the entire trace. Thus, the power spectrum (or autocorrelation) determining the deconvolution operator is approximately that of the evolved seismic waveform that actually illuminates the target. This also means that that operator is not the best operator for structures above or below the target, and these suffer systematic distortions, to be discussed. The practice of a target-focused deconvolution design is not universal. Another common strategy is to use as much of the trace as possible for the design, with the assumption that this results in some sort of average operator which will treat all stratigraphic levels similarly. This assumption will be seen to be incorrect. A major concern when specifying the design window is how large it should be. Obtaining the best resolution in the target window argues for a temporally small window; however, too small a window will give a distorted power spectrum that may be influenced more by the window than by the local wavefield. Rules of thumb exist relating window size to operator length, but these have considerable variation and are often ignored.

Attenuation is not the only time-variant problem to influence deconvolution operator design. Source-generated "noise" such as ground roll (Rayleigh waves) and refractions associated with the first breaks are also a problem. The spectrum of these wave modes is usually quite different from that of the downgoing wavefield that is illuminating subsurface targets. Allowing them into the operator design window is a major source of phase and amplitude errors. The simplest way of dealing with these modes is to define the design window to exclude them. This is relatively easy to do for refractions and first breaks by just starting the window after them; however, ground roll is dispersive and often spans a large time zone that overlies the reflections of interest. Ground-roll effects can often be reduced by *f–k* filtering or specially designed nonlinear codes that recognize unique features of Rayleigh waves. These will not be discussed in this book.

5.6.1 Stationary and Nonstationary Trace Models

At the beginning of this chapter (Section 5.1), we introduced the convolutional model of a seismic trace as the basis for stationary deconvolution. This model is simply written as

$$s(t) = (w \bullet r)\,(t) = \int_{-\infty}^{\infty} w(t-\tau) r(\tau)\, d\tau, \tag{5.41}$$

where, as before, $w(t)$ is the wavelet, $r(t)$ is the reflectivity, $s(t)$ is the seismic trace, and we have neglected additive noise to simplify the equations and the discussion. Also, in Section 4.7.4, we introduced the construction of nonstationary seismograms via the Q matrix, which was a generalization of a convolution matrix. The Q matrix idea can be extended to the continuous case to give an expression similar to Eq. (5.41),

$$s(t) = \int_{-\infty}^{\infty} a(\tau, t-\tau) r(\tau)\, d\tau. \tag{5.42}$$

Here we specify that $a(\tau, t-\tau)$ represents the anelastic attenuation process for an initial Dirac delta impulse. Comparing Eqs. (5.41) and (5.42) shows that $w(t-\tau)$ has been replaced by $a(\tau, t-\tau)$. The fact that a depends on both τ and $t-\tau$ means that Eq. (5.42) is a nonstationary convolution. If the constant-Q model of attenuation is invoked (Section 4.7.4), then a is given by

$$a(t,\tau) = \int_{-\infty}^{\infty} \alpha(t,f) e^{2\pi i f t}\, df, \tag{5.43}$$

with $\alpha(t,f)$ given by

$$|\alpha(t,f)| = e^{-\pi |f| t/Q}, \tag{5.44}$$

with the phase of α being defined by the minimum-phase condition (see Margrave et al. (2011) for more discussion). Defined in this way, Eq. (5.42) assumes a perfect Dirac delta (i.e., impulsive) source and so is not quite parallel to Eq. (5.41), which explicitly allows any wavelet but omits attenuation. To incorporate an arbitrary source wavelet, we first

Fourier transform Eq. (5.42) to get (Margrave, 1998)

$$\hat{s}(f) = \int_{-\infty}^{\infty} \alpha(t,f) r(t) e^{-2\pi i f t}\, dt. \tag{5.45}$$

Now, in the Fourier domain an arbitrary wavelet can be applied by multiplication, so we modify Eq. (5.45) to

$$\hat{s}(f) = \hat{w}(f) \int_{-\infty}^{\infty} \alpha(t,f) r(t) e^{-2\pi i f t}\, dt. \tag{5.46}$$

This is the final form of our nonstationary trace model. It is convenient to leave it expressed in the frequency domain, where it is mathematically simpler. However, if transformed to the time domain it becomes, in the discrete case, the Q matrix applied to the reflectivity formulation discussed in Section 4.7.4. Similarly, the stationary trace model of Eq. (5.41) becomes, in the discrete case, the convolution matrix applied to a reflectivity. The reader is referred to Figures 4.19 and 4.20 for further understanding. The time-domain equivalent of Eq. (5.46) can be written symbolically as

$$s(t) = (w \bullet [a \odot r])\,(t), \tag{5.47}$$

where $a \odot r$ symbolizes the nonstationary convolution of Eq. (5.42) and we note that the operations $w\bullet$ and $a\odot$ do not commute. For sampled signals, we write the matrix equivalent to Eq. (5.47) as

$$\underline{s} = \underline{\underline{W}}\,\underline{\underline{A}}\,\underline{r}, \tag{5.48}$$

where $\underline{\underline{W}}$ is a Toeplitz convolution matrix for w, $\underline{\underline{A}}$ is a non-Toeplitz Q matrix representing $a(\tau, t-\tau)$, and $\underline{r}$ is a column vector containing the reflectivity series in time. The Q matrix of Figure 4.20 is the matrix product $\underline{\underline{Q}} = \underline{\underline{W}}\,\underline{\underline{A}}$. Formally, the inverse of Eq. (5.48) is

$$\underline{r} = \left(\underline{\underline{W}}\,\underline{\underline{A}}\right)^{-1} \underline{s}, \tag{5.49}$$

which is equivalent to

$$\underline{r} = \underline{\underline{A}}^{-1}\,\underline{\underline{W}}^{-1}\,\underline{s}. \tag{5.50}$$

Equation (5.50) says that the wavelet inverse must be applied before the attenuation inverse if they are treated separately. Gabor deconvolution essentially estimates the combined inverse $\left(\underline{\underline{W}}\,\underline{\underline{A}}\right)^{-1}$, although in the Gabor domain.

It is a virtue of the nonstationary trace model that it becomes identical to the stationary model in the limit of no attenuation. In this case, $\alpha(t,f) = 1$ and hence Eq. (5.46) reduces to $\hat{s}(f) = \hat{w}(f)\hat{r}(f)$, which is just the convolutional model in the frequency domain. Alternatively, in the stationary limit of the time-domain formulation, $\underline{\underline{A}}$ becomes the identity matrix and this time Eq. (5.48) reduces to the convolutional model.

5.6.2 Applying Stationary Deconvolution to Nonstationary Traces

It is instructive to examine the behavior of stationary deconvolution on nonstationary synthetic traces. Figure 5.19 shows four synthetic traces, two stationary and two nonstationary,

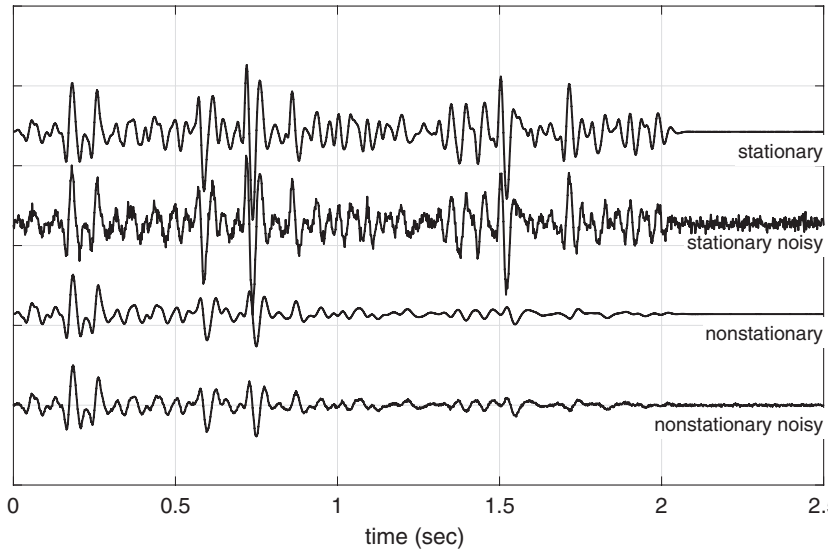


Figure 5.19 Stationary and nonstationary traces corresponding to the same reflectivity. The stationary traces are identical to those used in previous examples, while the nonstationary ones were made with a Q matrix using $Q = 50$. Noise was added to both using a signal-to-noise ratio of 4 but, for the nonstationary trace, the ratio was determined from 1 to 1.5 s. Since the signal has decayed in this zone, the added noise is also weaker to get the same signal-to-noise ratio. These traces were created by the script `deconcode/makeQsynthetic.m`.

both with and without noise. Code Snippet 5.6.1 illustrates the creation of these traces. The stationary traces are the same ones as used in the previous examples in this chapter, while the nonstationary traces were created with the same reflectivity and wavelet but with a Q matrix (Section 4.7.10) for $Q = 50$. As discussed in the previous section, the Q matrix approach is equivalent to Eq. (5.46). In this case, `qmatrix` has been run with the fifth input set to 1, which causes the nonstationary phase delays (the drift) to be determined with respect to the Nyquist frequency rather than the much higher well-logging frequency. This means that it is theoretically possible for a deconvolution method to estimate and remove these delays. Careful examination of these traces shows many of the features described in Chapter 4. The stationary and nonstationary seismograms are very similar in the first 200 ms or so but then become progressively very different. The stationary seismogram shows essentially constant average amplitude and dominant frequency, while the nonstationary result shows progressive loss of both amplitude and higher frequencies. The careful eye will also notice progressive time delay (drift) in the nonstationary case. The addition of random noise with a specific signal-to-noise ratio is a simple matter in the stationary case, but in the nonstationary case the signal-to-noise ratio must be specified in a specific time window. The rms amplitude is a measure of signal strength computed by $\sqrt{\sum_k s_k^2}/N$, where the sum is taken over a temporal window and N is the number of samples in the window. For a window size of perhaps 100 ms, the stationary trace will show a nearly constant rms amplitude, while the nonstationary one will show a progressive rms amplitude decay. If

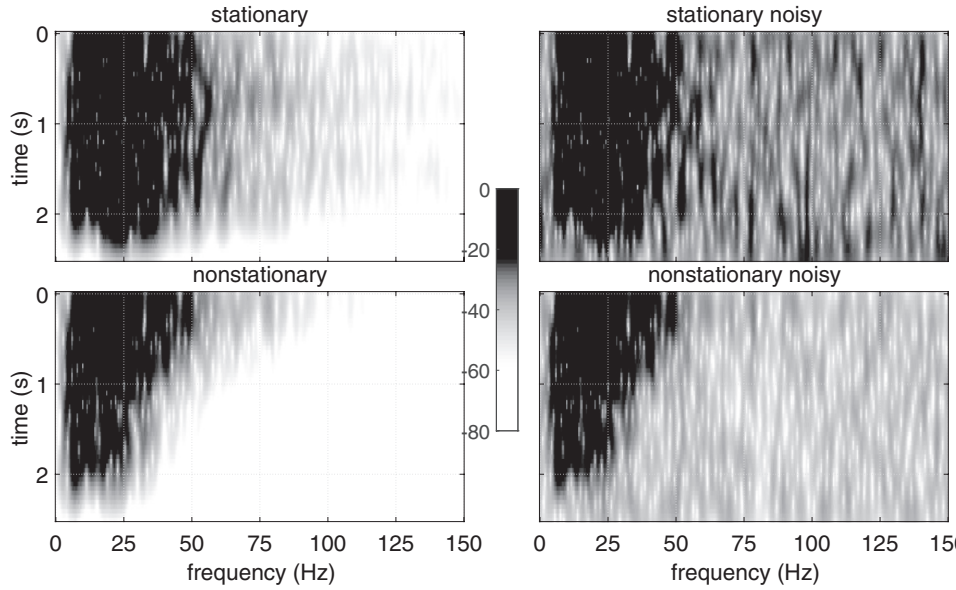


Figure 5.20 Gabor amplitude spectra of the traces in Figure 5.19. The Gaussian window width (standard deviation) was 0.3 s and the window increment was 0.05 s. Gray-level shading indicates amplitude in decibels, and in the center is a "colorbar" that applies to all four spectra. See the script `deconcode/makeQsynthetic_all.` for the computation of these spectra.

the additive noise has constant rms amplitude, then specifying the signal-to-noise ratio to be 4 in the time window 1–1.5 s means that, for the nonstationary trace, the signal-to-noise ratio will be greater than 4 for $t < 1$ and less than 4 for $t > 1.5$, while for the stationary signal the signal-to-noise ratio will not vary.

Figure 5.20 shows the Gabor amplitude spectra of the traces in Figure 5.19. The Gabor spectrum (Section 3.7.1) is essentially just a collection of Fourier spectra computed in local windows that are defined to span the length of the trace. In this case, the windows were Gaussians with a standard deviation of 0.3 s, and the window centers were spaced 0.05 s apart and spanned the length of the traces. There is considerable flexibility in these window parameters and it is instructive to recreate these with different choices for `twin`, which is the window standard deviation. The important thing is to ensure that `tinc`, the window spacing, is considerably smaller than `twin`. Comparing the stationary and nonstationary spectra in the noise-free case reveals the essential difference. The stationary Gabor spectrum shows very little large-scale temporal variation, while the nonstationary case shows a well-defined trend of decreasing spectral width with increasing time. In fact, if these spectra were to be smoothed in the temporal direction, there would be almost no variation in the stationary spectrum. The spectral shape in the nonstationary case is the expected behavior of the constant-Q model of attenuation described in Section 4.7.3. In fact, if Q is not a function of time, then it is expected that the amplitude spectrum will be essentially

Code Snippet 5.6.1 This illustrates the creation of stationary and nonstationary seismograms with and without additive random noise. Lines 1–6 define some parameters and line 8 creates a "random" reflectivity using π as a seed for the random number generation, ensuring that the same reflectivity is generated each time. A minimum-phase wavelet is generated on line 13 and included in the creation of the Q matrix on line 14. The fifth input in *qmatrix* determines that phase delays will be with respect to the Nyquist frequency, not the well-logging frequency. The nonstationary seismogram is created on line 15 and the stationary one on line 16. Noisy versions of both are created on lines 17–19. See Figures 5.19, 5.20, and 5.22 for an exploration of the results.

```
1  dt=.002;%time sample rate
2  Q=50;%Q value
3  tmax=2;%max time for reflectivity
4  fdom=20;%dominant frequency of wavelet
5  tlen=.5*tmax;%length of wavelet (this is overkill)
6  s2n=4;%signal-to-noise ratio
7  %change the last argument in reflec (currently pi) to any
8  %other number to get a different reflectivity
9  r=reflec(tmax,dt,.1,3,pi);
10 tmax=tmax+.5;%pad out a bit
11 t=(0:dt:tmax)';%t coordinates
12 r=pad_trace(r,t);%pad r with zeros
13 [w,tw]=wavemin(dt,fdom,tlen);%the wavelet
14 qmat=qmatrix(Q,t,w,tw,1);
15 sq=qmat*r;%nonstationary synthetic
16 s=convm(r,w,0);%stationary synthetic
17 sn=s+rnoise(s,s2n);%add some noise to stationary
18 iz=near(t,1,1.5);%zone defining noise strength for nonstationary
19 sqn=sq+rnoise(sq,s2n,iz);%add some noise to nonstationary
```

End Code

deconcode/makeQsynthetic.m

constant along any of the curves defined by $\tau f =$ constant, where τ is the Gabor window-center time and f is the frequency. These curves are hyperbolas whose asymptotes are the τ and f axes. The addition of noise makes the temporal variation of the nonstationary trace slightly harder to discern, but it is still there. The colorbar has been set such that attenuation levels below -80 dB are not visible. This is because the Q exponential decay leads to very large attenuation values. In decibels, the attenuation is

$$a_{\text{dB}} = 20\log_{10} e^{-\pi ft/Q} = -20\log_{10}(e)\pi ft/Q \approx -27.3ft/Q. \tag{5.51}$$

Figure 5.21 shows these attenuation levels for three different values of Q. These are not the total attenuation, because the wavelet is not considered here. Attenuation levels below -80 dB are generally not recoverable even for high-quality synthetic data. This means that, even after a high-quality nonstationary deconvolution, the signal bandwidth will always be time-variant and generally decreasing with increasing time.

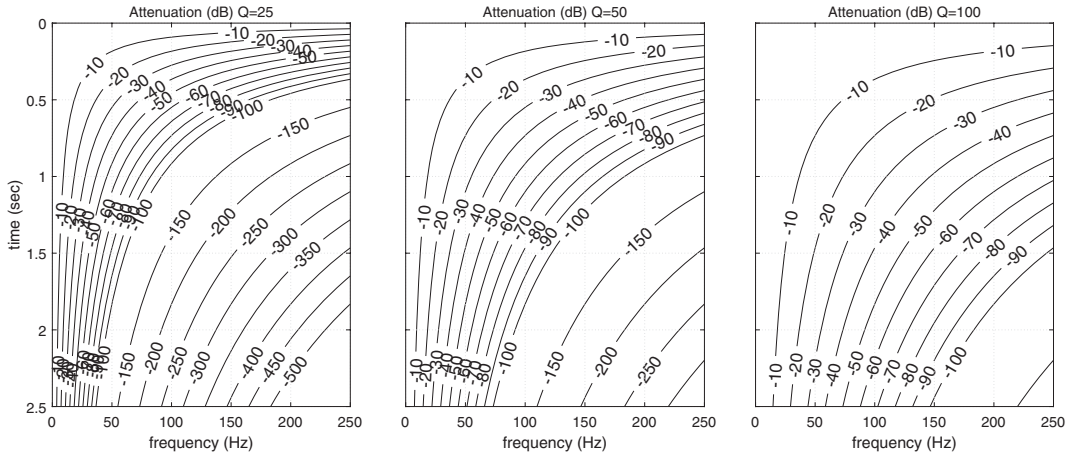


Figure 5.21 Attenuation levels in decibels are shown for three different values of Q as determined by Eq. (5.51). The levels are constant along the hyperbolas defined by $tf = \text{constant}$. The total attenuation, as seen in Figure 5.20, also includes the wavelet. Attenuation levels below −80 dB are generally not recoverable. See the script `deconcode/makeQsynthetic_all.` for the creation of this figure.

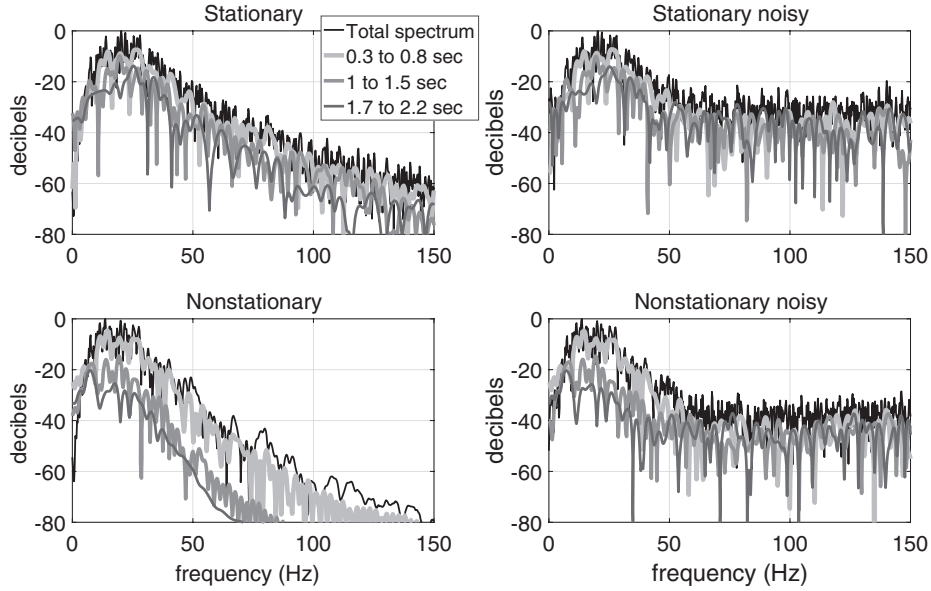


Figure 5.22 This is an alternate way to study the stationary and nonstationary spectra of the traces in Figure 5.19. For each of the four traces, Fourier spectra in three time windows and the total spectrum are shown.

Figure 5.22 shows another way to view the temporal dependence of spectra. Here are shown the total Fourier spectrum of each trace together with three local spectra from three time windows. This view is more helpful than the Gabor spectra in understanding the performance of stationary deconvolution on these traces. Considering the noise-free

case, suppose that the 1–1.5 s window is chosen for the operator design and that the frequency-domain algorithm is used. Then the operator will be such that it flattens (i.e., whitens) this spectrum, yet it is applied to all times, and hence all the noise-free spectra of this figure. In the stationary case, the distinctive feature is that all of the spectra have essentially the same shape. So, a multiplicative operator that flattens one of them will flatten them all. For the nonstationary case, the story is quite different. The 1–1.5 s window shows a spectral decay that is intermediate between that of the other two windows. This means that the operator which whitens the middle window will overcorrect the shallow window and undercorrect the deeper one. Also, notice that the total spectrum has a shape most similar to that of the shallow window and is therefore not some simple average spectrum. This is because the shallow window, having stronger amplitudes, dominates the total spectrum. It follows that if the entire trace is used for operator design, then the shallow part will be approximately whitened and the deeper part, which is most likely where the zone of interest lies, will be left underresolved.

Code Snippet 5.6.2 This code performs a stationary deconvolution of the noise-free synthetics created in Code Snippet 5.6.1 and shown in Figure 5.19. The stationary deconvolution is done with *deconf* using a time window of 1–1.5 s for operator design. Lines 3 and 4 define the design window and line 5 computes a window function, `mw`, to taper the design trace (see *mwindow* for more information). Lines 6–9 establish the `stab` factor and `fsmo` the frequency smoother. A much larger `stab` value is needed in the nonstationary case owing to the much greater attenuation levels. The stationary trace is deconvolved on line 10 and the nonstationary one on line 11. Note the use of `mw` in the second input argument. Lines 12 and 13 balance the deconvolved traces to the reflectivity for better display. Finally, lines 15 and 16 measure crosscorrelations and apparent residual phase. The deconvolved traces appear in Figure 5.23.

```
1  %deconvolve the Q synthetic noise-free traces.
2  %Be sure to run makeQsynthetic before this
3  t1=1;t2=1.5;%define the design window
4  ind=near(t,t1,t2);%indices of design window
5  mw=mwindow(length(ind),40);%window function
6  stab=.00001;%decon stab stationary
7  stabn=.001;%decon stab nonstationary
8  fsmo=5;%frequency smoother in Hz
9  nsmo=round(fsmo/t(end));%fsmo in samples
10 sd=deconf(s,s(ind).*mw,nsmo,stab,1);%stationary case
11 sqd=deconf(sq,sq(ind).*mw,nsmo,stabn,1);%nonstationary case
12 sd=sd*norm(r(ind))/norm(sd(ind));%amplitude balance
13 sqd=sqd*norm(r(ind))/norm(sqd(ind));%amplitude balance
14 ncc=40;%number of correlation lags
15 [x,strstat]=maxcorr_ephs(r,sd,ncc);%measure cc and phase
16 [x,strnon]=maxcorr_ephs(r,sqd,ncc);%measure cc and phase
```

End Code

deconcode/deconfQsynthetic .m

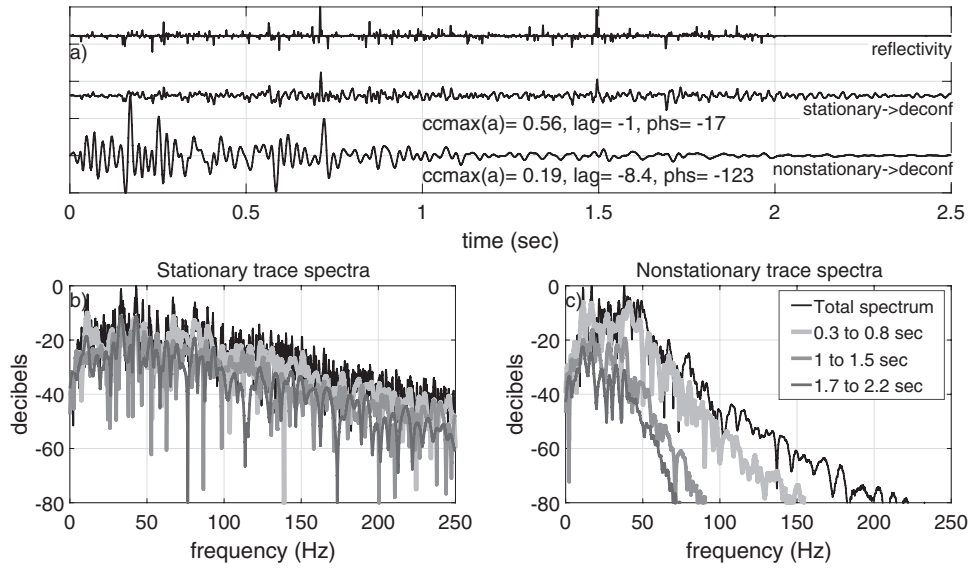


Figure 5.23 (a) The results of a stationary deconvolution (see Code Snippet 5.6.2) of the noise-free traces in Figure 5.19. The deconvolution design window was from 1 to 1.5 s. Annotated beneath each deconvolved trace are comparison statistics for the reflectivity measured for the design window. (b) Time-variant spectra for the stationary trace after deconvolution; (c) similar spectra for the nonstationary trace.

Code Snippet 5.6.2 illustrates the computations necessary to deconvolve the nonstationary noise-free synthetic and also, for comparison, the corresponding stationary trace. The strategy employed here is to design the operator over a restricted time interval, which should span the zone of interest, the idea being that this operator will then be optimal for the zone of interest and accepting that it will be less so elsewhere. The stationary synthetic is also deconvolved with an operator from this same time interval (from 1 to 1.5 s) for a fair comparison. The results are shown in Figure 5.23, and it is apparent that the nonstationary trace has led to a very poor result. The maximum crosscorrelation, even though it was restricted to the design gate, is very low and the residual phase is very large. There are obvious amplitude distortions all along the trace. The deconvolved traces were balanced to the reflectivity over the design gate to have the same rms amplitude and, at earlier times, the amplitudes are too strong while at later times they are too weak. A gain correction can help somewhat but will not improve the crosscorrelation or residual phase. The stationary trace has deconvolved fairly well, although a much better result can be obtained with a larger design window, something that is not true for the nonstationary trace. The spectra of the stationary trace after deconvolution are all very similar, showing that the essential stationarity has not been disturbed. In contrast, the nonstationary trace still shows strong nonstationarity after deconvolution, although the spectra have changed shape. The spectrum from the design window is roughly flattened out to 70 Hz or so, while the shallow time window shows some evidence of overwhitening (the spectral amplitudes increase with increasing frequency) and the deeper window is underwhitened. This result is typical of attempts to deconvolve a nonstationary signal with a stationary method; however, there

are a number of other possible strategies. Exploring these is left to the exercises, as is the deconvolution of the noisy synthetics.

Exercises

5.6.1 Construct the nonstationary traces used in this section, and deconvolve the noise-free trace with your choice of stationary methods but using the entire trace for the operator design. Plot your results and describe them.

5.6.2 Construct the nonstationary traces used in this section and use *tgain* to apply a gain correction to the noise-free trace before deconvolution. Determine the *tgain* parameter `n` by trial and error. Then deconvolve the gained trace with your choice of stationary methods but using the entire trace for the operator design. Plot your results and describe them.

5.6.3 Repeat Exercise 5.6.2 using the noisy nonstationary synthetic and compare the results.

5.6.4 Using the stationary traces from this section, deconvolve them with the same design gate as used in the text (1–1.5 s) and then again with two progressively larger gates. Plot your results and describe them. How does the size of the design gate affect the results? Repeat this with the corresponding nonstationary traces. Do you see the same dependence? Explain.

5.6.5 Construct the stationary and nonstationary traces used in this section and then develop a code to compute the rms amplitude over a specified time window. Compute another noise-free nonstationary trace for $Q = 100$. Then compute and plot the rms amplitude of the noise-free stationary traces and the two noise-free nonstationary traces for $Q = 50$ and $Q = 100$ for window center times that span the length of the traces. Use 0.1 s boxcar windows. Discuss your results.

5.6.6 Construct the nonstationary traces used in this section and, following the example in the script `deconcode/makeQsynthetic_all`, plot the Gabor amplitude spectra of the noise-free and noisy traces as in Figure 5.20. Develop code to plot on top of each of these spectra at least three hyperbolic curves satisfying $\tau f = c$, where c is a constant. Choose suitable values for c such that your curves fall on top of the Gabor spectra. Discuss your results.

5.6.3 Gabor Deconvolution

Gabor deconvolution (Margrave et al., 2011) is a direct extension of frequency-domain deconvolution to the nonstationary setting. (At this time, it is recommended to review Section 5.3.1 if this is not familiar.) Glossing over many details, the essence of the frequency-domain method is that we (1) Fourier transform the signal, (2) divide the spectrum of the signal by its smoothed self, and (3) perform an inverse Fourier transform to recover the deconvolved signal. This vast simplification omits many details, including the very important considerations of how to smooth the spectrum and how to compute the

phase; nevertheless, it is a useful viewpoint. Gabor deconvolution proceeds in a very similar manner, where we (1) Gabor transform the signal, (2) divide the Gabor spectrum by its smoothed self, and (3) perform an inverse Gabor transform to recover the deconvolved signal. The details of the Gabor process are of essential importance, but the larger picture emphasizes the similarities between Gabor deconvolution and the stationary methods. Before describing the algorithmic details, consider why the `deconf` process works so well on a stationary trace. Most fundamentally, this is because the convolutional model (see Section 4.7.2), $s(t) = (w \bullet r)(t)$, which describes the stationary trace, is *factorized* by the Fourier transform. This means simply that, in the frequency domain, the convolutional model is $\hat{s}(f) = \hat{w}(f)\hat{r}(f)$. In a similar fashion, the application of the Gabor transform to the nonstationary trace model accomplishes the approximate factorization

$$\hat{s}_g(t_j,f) \approx \hat{w}(f)\alpha(t_j,f)\hat{r}_g(t_j,f), \tag{5.52}$$

where $\hat{s}_g(t_j,f)$ is the Gabor transform of the trace, $\hat{w}(f)$ is the Fourier transform of the wavelet, $\alpha(t_j,f)$ is the attenuation function appearing in the nonstationary trace model, and $\hat{r}_g(t_j,f)$ is the Gabor transform of the reflectivity. This approximate factorization is derived in Section 3.7.3, while here we mention its intuitive justification. Suppose that the attenuation function is not too severe and that we apply a Gaussian window to some position, t_j, on the seismic trace, where the window width is somewhat larger than the expected wavelet. Then, within this window, we can expect the convolutional model to approximately apply. After a Fourier transform, the effective wavelet spectrum will be the product $\hat{w}(f)\alpha(t_j,f)$, and the reflectivity spectrum within the window is $\hat{r}_g(t_j,f)$. The product of these two terms is the local convolutional model but it is also the factorization of Eq. (5.52).

There are two other key assumptions about the nature of $r(t)$ and $w(t)$ that enable the success of stationary deconvolution on stationary traces. Concerning the reflectivity, the whiteness assumption implies that smoothing the reflectivity spectrum will reduce it to a constant; therefore, smoothing the trace spectrum eliminates the reflectivity contribution. This is why dividing the trace spectrum by its smoothed self will isolate the reflectivity (to within a constant scale factor). We make a similar assumption about the Gabor spectrum of the reflectivity, that a suitable smoothing process will reduce it to a constant. Concerning the wavelet, since the spectral smoothing is applied to the amplitude spectrum only in frequency-domain deconvolution, the phase spectrum of the wavelet must be determined by some other means. The minimum-phase assumption is therefore invoked to allow the phase spectrum of the wavelet to be estimated from its amplitude spectrum. In the nonstationary case, we still assume that the wavelet emitted by the source is minimum phase and we further assume that the attenuation process is also minimum phase. Then it follows that the *propagating wavelet*, $\hat{w}(f)\alpha(t,f)$, is always locally minimum phase. Therefore it will be possible to estimate the wavelet's phase by a Hilbert transform over frequency at constant time of the estimated Gabor amplitude spectrum.

Having stressed the link between frequency-domain deconvolution and Gabor deconvolution, we proceed to detail the Gabor algorithm. Equation (5.52) can be solved for the

Gabor transform of the reflectivity as

$$\hat{r}_g(t_j,f) = \frac{\hat{s}_g(t_j,f)}{\hat{w}(f)\alpha(t_j,f)}. \tag{5.53}$$

While symbolically true, this result cannot be implemented directly, because both $\hat{w}(f)$ and $\alpha(t_j,f)$ are unknown. We postulate that a smoothing process can be developed such that the smoothed Gabor magnitude spectrum of the trace can estimate the magnitude of the product of these unknowns. That is,

$$\overline{\left|\hat{s}_g(t_j,f)\right|} \approx \left|\hat{w}(f)\alpha(t_j,f)\right|, \tag{5.54}$$

where the overbar indicates the as yet unspecified smoothing process. If the propagating wavelet, $\hat{w}(f)\alpha(t_j,f)$, is minimum phase, then the phase estimate follows as

$$\phi_g(t_j,f) = \mathcal{H}\left[\overline{\left|\hat{s}_g(t_j,f)\right|}\right], \tag{5.55}$$

where the Hilbert transform is to be accomplished over f at constant t_j. Then Eq. (5.53) can be implemented as

$$\hat{r}_g(t_j,f) = \hat{s}_g(t_j,f)\left[\frac{e^{-i\phi_g(t_j,f)}}{\overline{\left|\hat{s}_g(t_j,f)\right|}}\right], \tag{5.56}$$

where the term in the square brackets is the Gabor deconvolution operator and an inverse Gabor transform will recover the time-domain reflectivity estimate. Just as in the stationary methods, the deconvolution operator, $e^{-i\phi_g}\overline{\left|\hat{s}_g\right|}^{-1}$, depends entirely on the amplitude spectrum of the nonstationary trace and not on the phase.

It remains to describe several spectral smoothing algorithms that have proven useful. Each smoothing method will result in a different result for $\overline{\left|\hat{s}_g(t_j,f)\right|}$, but the Gabor deconvolution is formally accomplished by Eq. (5.56) in each case. At this time, we have no general mathematical theory describing an optimal method to estimate $\left|\hat{w}(f)\alpha(t_j,f)\right|$ from $\left|\hat{s}_g(t_j,f)\right|$. It may be that spectral smoothing is not the best method, but it has proven successful in frequency-domain deconvolution; however, as discussed in Section 5.3, smoothing approaches are inherently biased, and that bias extends to the nonstationary context.

To better understand these issues, Figure 5.24 illustrates the three factors in the Gabor factorization of the nonstationary trace as shown in Eq. (5.52). From left to right are shown the reflectivity, $r(t)$, its Gabor amplitude spectrum, $\left|\hat{r}_g(t_j,f)\right|$, the magnitude[6] of the attenuation function, $\alpha(t_j,f)$, and the amplitude spectrum of the source wavelet, $w(t)$. The latter was constructed by simply replicating the Fourier transform of the wavelet along the time axis. The Gabor amplitude of the reflectivity, $\left|\hat{r}_g(t_j,f)\right|$, shows essentially random fluctuations near a constant value and is characterized by the term "white" with reference to the spectrum of white light.[7] The attenuation function shows the basic shape of the exponential $e^{-\pi ft/Q}$ inherent in the constant-Q model. As remarked previously, the level-lines

[6] Magnitude and amplitude are treated synonymously here.

[7] We are amused by the paradox that our display of a white spectrum is black.

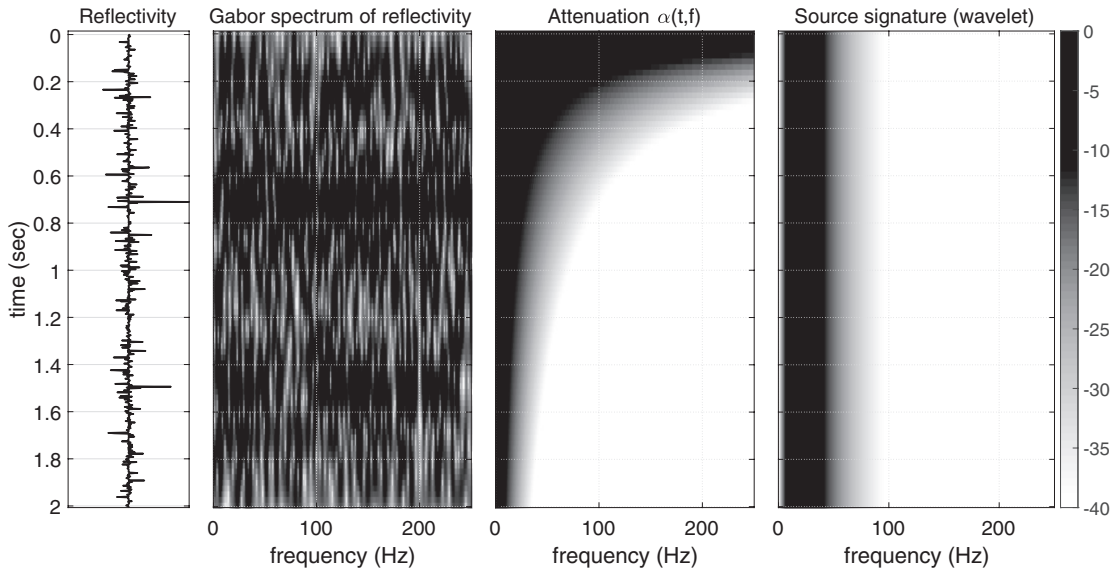


Figure 5.24 The three factors of the nonstationary trace model in the Gabor domain (see Eq. (5.52)). The reflectivity and other parameters are the same as used in Figure 5.19. This reflectivity is shown again at left for comparison with its Gabor amplitude spectrum, which used Gaussian windows with a standard deviation of 0.15 s and a spacing of 0.025 s. The attenuation function was simply computed directly from its formula, and the source signature is just the Fourier amplitude spectrum of $w(t)$ repeated at all times. The gray scale for all panels is shown at the far right and is in decibels. This figure was created by `deconcode/gabordecon_all`.

(i.e., contours) of this function are the family of hyperbolas $tf =$ constant, and that shape is apparent in the figure. The amplitude spectrum of the source wavelet is independent of time and so is displayed without temporal variation.

The product of the three time–frequency surfaces of Figure 5.24 forms our model for the Gabor spectrum of the nonstationary trace as expressed by Eq. (5.52). This is illustrated in Figure 5.25. The first panel on the left shows the product of the attenuation and source signatures from Figure 5.24. We call this product the propagating wavelet because the inverse Fourier transform at any time gives the wavelet at that time as predicted by the nonstationary convolution theory. The product of all three panels of Figure 5.24 gives the Gabor spectrum model of Figure 5.25, and displayed next to it is the actual Gabor amplitude spectrum of the nonstationary trace shown at the far right. The model and actual Gabor spectra are extremely similar, although not identical. This shows that the Gabor factorization of the nonstationary trace model, while approximate, is very, very good.

In Figure 5.25, we can visualize the requirements for a good spectral smoothing process. The actual Gabor spectrum must be processed in such a way that the propagating-wavelet spectrum is the result. Clearly, the latter is much smoother than the former, so perhaps an overt convolutional smoother will be of value. In frequency-domain stationary deconvolution, a 1D convolutional smoother was used, so perhaps a 2D convolutional smoother will be similarly useful here. In our MATLAB implementation, `gabordecon`, a 2D convolution

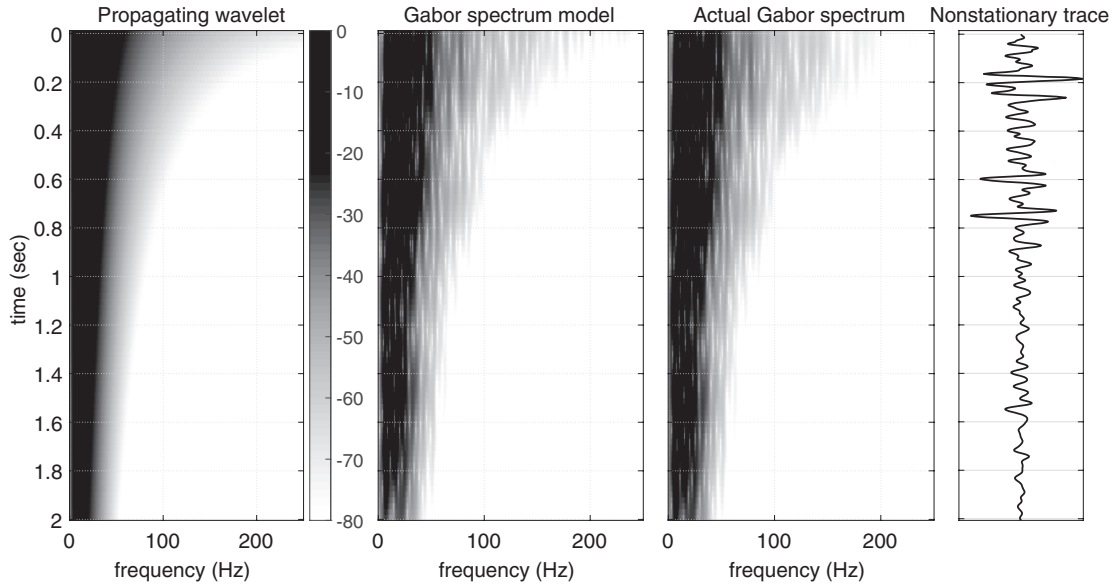


Figure 5.25 Equation (5.52) can be considered as a mathematical model for the Gabor spectrum of a nonstationary trace. The three factors involved were shown in Figure 5.24, and here these factors are shown as multiplied together. All spectra are the amplitude spectrum only. At left is the product $\hat{w}(f)\alpha(t_j, f)$, which is called the propagating wavelet. In the second panel is the Gabor spectrum model, which has all three terms of Eq. (5.52), and in the third panel is the actual Gabor spectrum. On the right is the nonstationary trace itself, in scale with the spectra. The gray scale for all spectra is shown on the left panel and is in decibels. This figure was created by `deconcode/gabordecon_all`.

with a 2D boxcar is one of two smoothing methods offered. This gives a basic capability that can be viewed as a direct extension of `deconf`. However, simple boxcar smoothing in the temporal direction can result in serious amplitude distortions in the reflectivity estimate. This happens when there is significant long-period (low-frequency) variation in the reflectivity on a scale exceeding the smoother size. Such considerations lead to the choice of a long temporal-smoother size, but this will usually lead to insufficient compensation for the attenuation. So, if the temporal smoother is too long, it will bias the estimate of the attenuation surface, but if it is too short, valuable relative amplitude information will be lost in the reflectivity estimate. It can be difficult to find a suitable compromise.

These considerations lead to a second smoothing procedure, called *hyperbolic smoothing*. In this process, we seek separate estimates of the attenuation function and the wavelet spectrum. The first step is to compute the average value of the Gabor amplitude spectrum, $|\hat{s}_g(t_j,f)|$, along hyperbolic curves tf = constant. Accomplished by the function `hypersmooth`, this process is inspired by the shape of the theoretical attenuation surface for a constant-Q value. This surface should be constant along these curves; however, if Q varies in time or space, or with any real data, this will not be precisely true. This hyperbolically smooth surface is taken as an estimate of the attenuation function. Then $|\hat{s}_g(t_j,f)|$ is divided by this estimate and a boxcar smoother is applied to the result. This is taken as an estimate of the source wavelet. The source wavelet estimate will only be independent

Code Snippet 5.6.3 Here is an example of running *gabordecon* on the noise-free nonstationary synthetic of Figure 5.19. Lines 1 and 2 establish the Gaussian window width and spacing. Lines 3 and 4 define the size of the boxcar smoother, while line 5 defines the `stab` factor. Line 6 sets the flag for boxcar smoothing and line 7 accomplishes the Gabor deconvolution. Lines 8–12 repeat this process for hyperbolic smoothing. Note the much longer temporal-smoother size for hyperbolic smoothing. The results are shown in Figure 5.26.

```
1   twin=.3;%Gaussian window width (standard deviation)
2   tinc=.05;%spacing between windows
3   tsmob=.3;%temporal smoother for boxcar
4   fsmob=5;%frequency smoother for boxcar
5   stab=0.0001;%Stability factor
6   ihyp=0;%flag for no hyperbolic
7   sgb=gabordecon(sq,t,twin,tinc,tsmob,fsmob,ihyp,stab);%boxcar smo
8   tsmoh=1;%temporal smoother for hyperbolic
9   fsmoh=5;%frequency smoother for hyperbolic
10  stab=0.0001;%stability factor
11  ihyp=1;%flag for hyperbolic
12  sgh=gabordecon(sq,t,twin,tinc,tsmob,fsmob,ihyp,stab);%hyprblc smo
```

End Code

deconcode/gabordcn.m

of time (as seen in Figure 5.24) if the temporal length of the boxcar is comparable to the trace length. Otherwise, some degree of temporal variation will remain, which tends to compensate for inaccuracies in the attenuation estimate.

As described, both boxcar smoothing and hyperbolic smoothing involve a boxcar smoothing process. The difference is that hyperbolic smoothing attempts to estimate and remove the attenuation surface before the boxcar smoothing. This allows a much larger temporal size for the boxcar smoother and potentially will lead to better relative amplitude preservation. In ordinary boxcar smoothing, the temporal size is typically about 0.2 or 0.3 s, while in hyperbolic smoothing, values greater than 1 s are common. For the frequency size, the same considerations as discussed in Section 5.3 are relevant, and typical values range between 5 and 15 Hz.

Code Snippet 5.6.3 illustrates the application of `gabordecon` to the noiseless nonstationary synthetic of Figure 5.19. Results for both smoothing methods are shown, and annotated beneath each result are statistics from three different time windows representing early, intermediate, and late times. In each case the crosscorrelation measures are with respect to the broadband reflectivity, and so higher values would result from properly band-limited comparisons. For both smoothing methods, the maximum crosscorrelation (`ccmax`) decays steadily from early to late, although the values from hyperbolic smoothing are greater than those for boxcar smoothing. Amplitude distortions are evident in both results, especially at the earliest times. There is some indication that the lag also increases with time, especially for the boxcar result. There has been no band-pass filter applied, and the time-variant band limiting results from the action of the stability constant, which suppresses whitening at a temporally decreasing maximum frequency. Imperfect though these

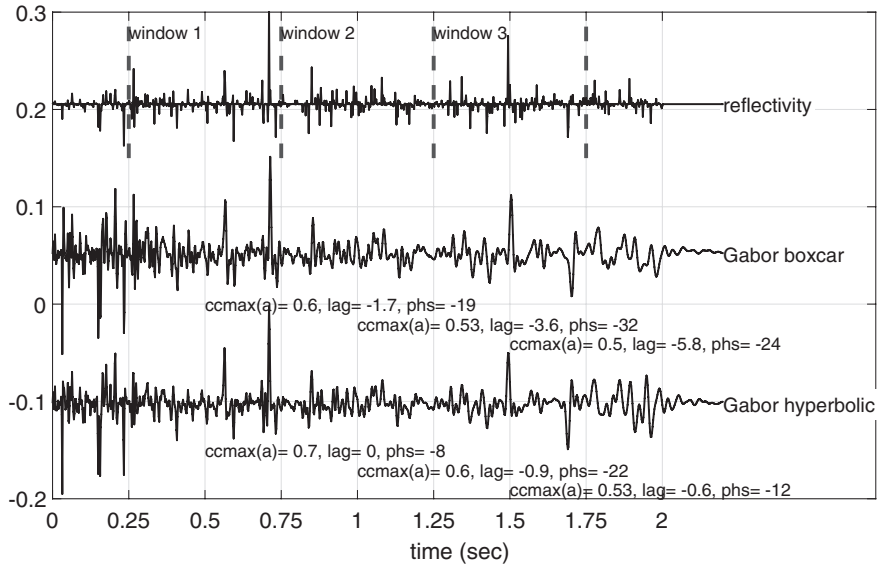


Figure 5.26 The results from Code Snippet 5.6.3 are shown for both boxcar and hyperbolic smoothing. No postdeconvolution filter has been applied. Annotated beneath each deconvolution are statistics for three different time windows. Each time window is centered where the annotation begins, and extends for 0.5 s. The windows are also indicated on the reflectivity. This figure was produced by `deconcode/gabordcn_all.m`.

results are, they are far better than what stationary deconvolution can achieve on the same signal (Figure 5.23).

Applying Gabor deconvolution to noisy traces runs into difficulties similar to those encountered in the stationary case. Any deconvolution process attempts to whiten the spectrum of the trace without consideration of noise levels. The deconvolution algorithms presented in this chapter have no mechanism to distinguish signal from noise, and so whiten all frequencies equally. In the nonstationary context, this whitening action takes place on the Gabor spectrum, which, as we have seen, is just a collection of windowed Fourier spectra. Code Snippet 5.6.4 deconvolves the noisy synthetic of Figure 5.19 and then applies a postdeconvolution band-pass filter to suppress the noise. The filtering action is done here by *hyperfilt*, which applies a time-variant band-pass filter whose filter parameters follow a hyperbolic path in the time–frequency plane. This is motivated by the fact that the constant-Q attenuation function is constant along such paths (see Figure 5.21). Assuming that the signal-to-noise ratio is therefore constant along such paths allows a determination of the filter parameters at one particular time to be extrapolated to all other times. For example, suppose we determine that the maximum signal frequency at time t_0 is the frequency f_0. Then, at some other time t_1, we assume the maximum frequency will satisfy $t_0 f_0 = t_1 f_1$, so that $f_1 = f_0 t_t / t_1$. It cannot be expected that this relationship will hold exactly, because the source wavelet's spectrum also changes the attenuation levels and because Q is likely not independent of time. For this reason, *hyperfilt* includes the ability to specify a "maximum maximum" frequency that may not be exceeded by this

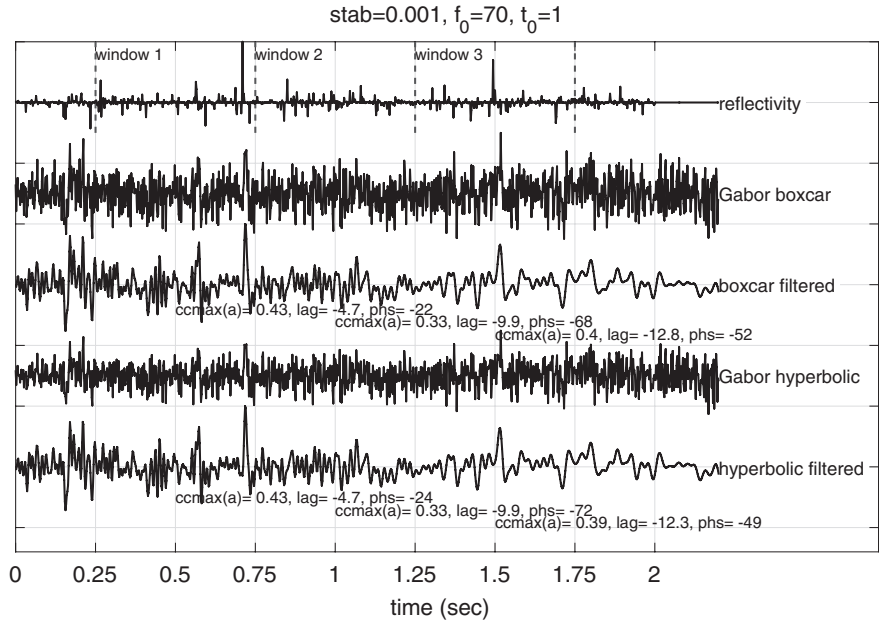


Figure 5.27 The results for Code Snippet 5.6.4, where the noisy synthetic is deconvolved and a further time-variant band-pass filter applied. Beneath the filtered results are crosscorrelation and phase statistics for three different windows in comparison with the reflectivity.

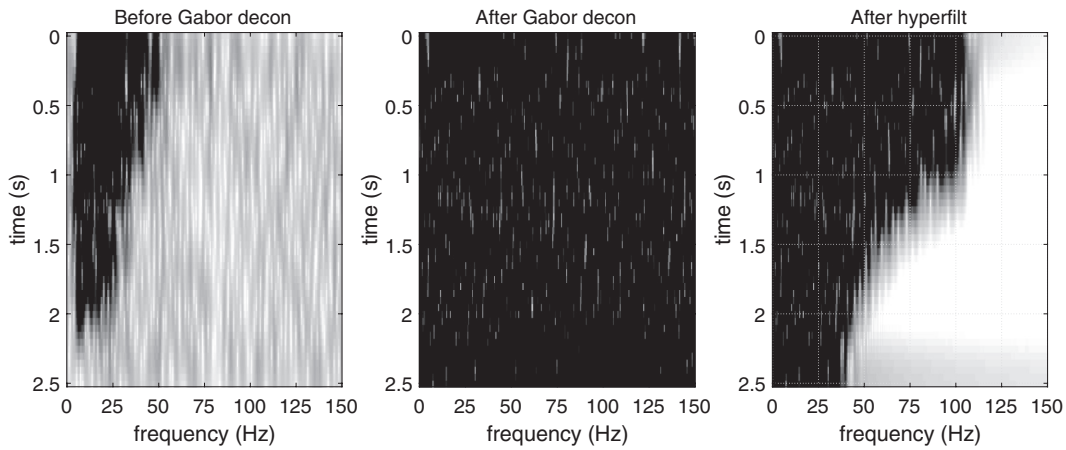


Figure 5.28 Gabor spectra for the results in Figure 5.27. At left is the spectrum before deconvolution, and in the middle is the spectrum after deconvolution. The black appearance of this middle spectrum means that the spectrum has been completely "whitened," even though much of it is noise dominated. At right is the spectrum of the deconvolved trace after the application of *hyperfilt*, which implements a time-variant filter whose filter parameters follow a hyperbolic path in the time–frequency plane. The hyperbolic path for the maximum frequency is discernible from 1 to 2.5 s. At earlier times the path has been overridden so as not to allow the maximum frequency to exceed 100 Hz. See `deconcode/gabordcn_all.m` for the creation of this figure.

Code Snippet 5.6.4 Similar to Code Snippet 5.6.3, except that we deconvolve the noisy synthetic of Figure 5.19 and this then requires a postdeconvolution filter for noise suppression. Lines 1–10 accomplish the deconvolution using both boxcar and hyperbolic filtering and then lines 11–14 apply a time-variant zero-phase low-pass filter using *hyperfilt*. Results are shown in Figure 5.27, and *hyperfilt* is described in the text.

```
1   twin=.3;%Gaussian window width (standard deviation)
2   tinc=.05;%spacing between windows
3   tsmob=.3;%temporal smoother for boxcar
4   fsmob=5;%frequency smoother for boxcar
5   stab=0.001;%Stability factor
6   ihyp=0;%flag for no hyperbolic
7   sgbn=gabordecon(sqn,t,twin,tinc,tsmob,fsmob,ihyp,stab);%boxcar smo
8   tsmoh=1;%temporal smoother for hyperbolic
9   ihyp=1;%flag for hyperbolic smoothing
10  sghn=gabordecon(sqn,t,twin,tinc,tsmob,fsmob,ihyp,stab);%hyprblc smo
11  %hyperbolic filtering
12  f0=70;t0=1;fmaxmax=100;fmaxmin=30;
13  sgbnf=filt_hyp(sgbn,t,t0,[0 0],[f0 10],[fmaxmax fmaxmin]);
14  sghnf=filt_hyp(sghn,t,t0,[0 0],[f0 10],[fmaxmax fmaxmin]);
```

End Code

deconcode/gabordcn_n.m

hyperbolic extrapolation. In Code Snippet 5.6.4, this value is set to 100 Hz on line 12. The filter action in this case is set as a time-variant low-pass filter with zero phase. The results of this computation are shown in Figure 5.27, where we see that, without the filtering, the results are essentially uninterpretable but, with the filtering, the most significant events can be discerned. The crosscorrelation statistics are, as expected, much lower than in the noise-free case but are still reasonable. Also of interest here is that there is almost no distinction between the two spectral smoothing techniques; however, this observation is for a very specific set of smoothing parameters. A better appreciation of the action of *hyperfilt* can come from examining Figure 5.28. In the rightmost panel, the hyperbolic trajectory of the maximum frequency can be observed between 1 and 2.5 s. At earlier times, the predicted maximum frequency exceeds the prescribed limit of 100 Hz, and this stops the hyperbolic extrapolation.

5.7 Chapter Summary

The methods of seismic deconvolution covered here are not the only possible ones, but they should give the reader an understanding of the nature and breadth of the topic. The chapter began with a presentation of the convolutional model that underlies all stationary deconvolution methods and which makes a practical solution possible. It was shown that a strictly physics-based convolutional model expresses the seismic trace as a wavelet convolved

with an impulse response, while the practical, industrial model substitutes reflectivity for impulse response. A first-order consideration of the difference between these two led to the discussion of gain correction as a preprocessing step before deconvolution.

The first deconvolution method considered was stationary spiking deconvolution performed in the frequency domain. This method exposes the fundamental difficulty of the problem most simply, as the frequency-domain trace is simply the product of spectra of the reflectivity and the wavelet, both of which are unknown. Thus the problem requires the separation of a measurement into the product of two unknown terms and is essentially unsolvable without further information. The possibility of a practical solution arises with the imposition of assumptions about the reflectivity and the use of a temporally short, minimum-phase wavelet. A random reflectivity has a rapidly fluctuating amplitude spectrum, while a temporally short wavelet has a smooth amplitude spectrum. Therefore, the amplitude spectrum of the trace is separated into smooth and rough parts, with the former being assigned to the wavelet and the latter to the reflectivity. It was emphasized that there are many ways to do this separation, and each possibility leads to a different solution. Once the separation of the amplitude spectra has been accomplished, the complete wavelet is realized through the minimum-phase assumption. Once the wavelet is known, the reflectivity arises from the complex-valued division of the trace spectrum by the wavelet spectrum. Thus, seismic deconvolution has two parts: wavelet estimation and then a mathematical deconvolution. The wavelet estimation part is much more important than the simple mathematical deconvolution and is where almost all of the difficulty lies.

Next, time-domain spiking deconvolution was presented and compared with the frequency-domain approach. It was emphasized that the two methods are essentially equivalent, given appropriate parameter choices, but the time-domain method is more common, while the frequency-domain approach is more intuitive. The presentation of the time-domain approach began with a discussion of the construction of a least-squares inverse filter, where it was shown that the minimum-phase assumption arises implicitly in the mathematical formulation and, given that, the inverse filter is determined by knowledge of only the autocorrelation of the forward signal itself. Thus, analogously to the separation of an amplitude spectrum into smooth and rough parts, in the time domain the signal autocorrelation must be separated into a wavelet part and a reflectivity part. It was shown that this amounts to attribution of the central lags (those around zero) to the wavelet and the other lags to the reflectivity. Given this, the least-squares inverse filter can be constructed and applied to the trace. The equivalence of the two methods was then discussed.

The concepts of prediction filters and prediction error filters were then introduced and it was shown that spiking deconvolution is essentially equivalent to a prediction-error filter of unit lag. Stated more plainly, the deconvolved seismic trace is identified with the unpredictable part, or prediction error, when a unit-lag prediction filter is applied. This leads to a generalization of stationary deconvolution into prediction operations with lags greater than unity. Such operations are called gapped deconvolutions and have historically been useful in attacking certain classes of long-delay multiples where the multiple is clearly separated from the wavelet. Another application proposes to use a short-gap operator, where the gap is greater than 1 but less than the expected wavelet length, to reduce the spectral whitening

effect of deconvolution and therefore eliminate the need for a postdeconvolution filter to reduce noise. It was shown that this practice can lead to a very nonoptimal embedded wavelet.

The final topic discussed was nonstationary deconvolution as specifically realized with the Gabor deconvolution method. First the essential nonstationarity of all seismic data was argued, and then the systematic errors that arise through the application of stationary deconvolution were examined. Gabor deconvolution was introduced as a direct extension of frequency-domain spiking deconvolution. The accommodation of nonstationarity is accomplished by replacing the Fourier transform with the Gabor transform, which is just a suite of temporally windowed Fourier transforms. The separation of the Gabor amplitude spectrum into wavelet and reflectivity is then accomplished by spectral smoothing much like the stationary frequency-domain method. In a series of examples, it was shown that nonstationary signals can be robustly deconvolved with the Gabor approach without knowledge of Q.

6 Velocity Measures and Ray Tracing

Exploration seismology is literally overflowing with *velocities*. Just to name a few, there are interval velocity, instantaneous velocity, apparent velocity, rms velocity, average velocity, mean velocity, stacking velocity, horizontal velocity, vertical velocity, phase velocity, group velocity, P-wave velocity, S-wave velocity, migration velocity, weathering velocity, and almost certainly others. This chapter is meant to bring some order to this chaos of velocities. For starters, there is a fundamental distinction between *physical velocities* and *velocity measures*. The former refers to velocities that are actually the speed at which some physical wave propagates. Examples are instantaneous velocity, P- and S-wave velocities, and phase and group velocities. On the other hand, velocity measures are typically quantities derived from data analysis that have the physical dimensions of velocity but are related to physical velocities in some indirect (and possibly unknown) fashion. Examples of velocity measures include average, mean, and rms velocities, interval velocity, stacking velocity, apparent velocity, and migration velocity. In contrast to physical velocities, it cannot generally be expected that a physical wave actually propagates at the speed of one of these velocity measures.

Using measured data for the analysis of velocity and the application to the data of corrections that depend upon velocity are fundamental to seismic processing. Velocity, together with the geometry of the seismic acquisition and the shapes of the reflectors, causes the characteristic traveltime shapes (such as hyperbolas) in the data. Spatial variations in velocity cause distortions from the simple, canonical shapes, and these distortions must be accounted for in processing. Sometimes velocity information can be obtained from supporting data such as well logs, core measurements, and geologic maps, but this information is inherently sparse with respect to the coverage of the seismic data itself. Ultimately, the only way to obtain the spatially dense velocity information required in processing is from the data itself. This is known as *velocity analysis* and is the source of some of the velocity measures such as stacking and migration velocities. The interpretation of the velocity measures such that they can be converted into measurements of physical velocities is called *velocity inversion* and is an ongoing problem. A careful discussion of the definitions of these measures is essential to gain understanding of the complexities of velocity inversion.

For simplicity, we consider the case of P-wave propagation in a heterogeneous Earth. (S-waves can be treated similarly.) A velocity measure will be defined by either relating it mathematically to the actual P-wave speed or by describing how it can be derived from seismic data.

6.1 Instantaneous Velocity: v_{ins} or Just v

Instantaneous velocity generally refers to the speed of propagation of seismic waves in the Earth. The word *instantaneous* refers to the local wave speed as the wave evolves from one instant of time to the next. The term can be applied to any wave type (P, S, surface, etc.), though here we specialize to P-waves for clarity.

Like most seismic "velocities," v_{ins} is not usually a vector quantity and so is not a velocity as physicists would use the term. Rather, it is a scalar which can be thought of as the magnitude of a velocity vector.

For practical seismic experiments, v_{ins} must be acknowledged to be a highly variable function of position in the Earth. In the most general case of anisotropic media, it also depends upon direction, but this will be ignored for now. It is not a function of time, as that would imply that the Earth's properties changed during the seismic experiment. Only rarely can it be assumed to be spatially constant, but often there is significant variation in one direction only. This is commonly the case in sedimentary basins, where velocity varies strongly with depth but only weakly in the horizontal direction.

Consider the instantaneous velocity function that is linear with depth z,

$$v_{\text{ins}} = v_0 + cz. \tag{6.1}$$

The *universal velocity function* is the name given to this function when $v_0 = 1800$ m/s and $c = 0.6\,\text{s}^{-1}$. Figure 6.1 shows this function for a depth range of 0 to 3000 m. The name originates from the days of analog computing (the 1950s and 1960s) when changing a velocity function was a considerable labor. For example, *normal moveout* was removed by a "computer" (who was a human being) who "traced the trace" (hence the name *trace* for a seismic recording) with a stylus that was connected to a pen by a lever arm driven by an

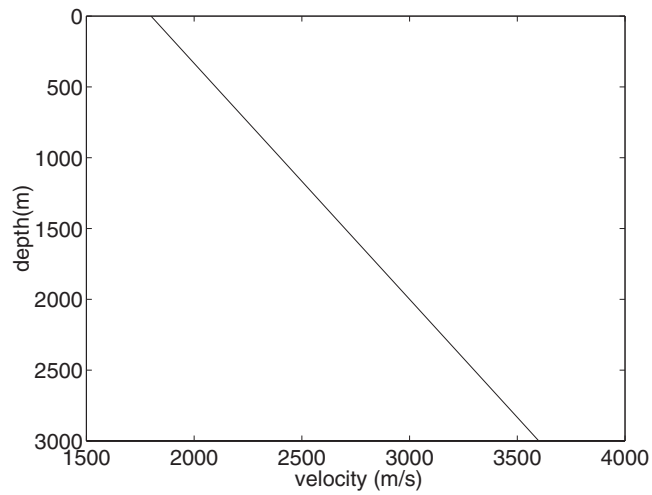


Figure 6.1 The universal velocity function is linear with depth.

eccentric cam. The pen would redraw the trace with normal moveout removed. However, the shape of the cam was determined by the velocity function and changing functions often meant going to the machine shop to cut a new cam. Thus the possibility of a universal function was quite attractive. And, it actually works quite well in many sedimentary basins.

6.2 Vertical Traveltime: τ

Given a complete specification of v_{ins} as a function of position, $v_{\text{ins}}(x, y, z)$, traveltimes over any path can be computed. Since there are infinitely many possible paths between any two points (of course, not all are Snell's law paths but all are possible routes for scattered energy), it is unreasonable to expect that there could be an unambiguous relation between traveltime and depth. However, if a special path is chosen, then such a relation can be developed. For this purpose, it is common to relate depth, z, to the traveltime along the vertical path from 0 to z. This *vertical traveltime* is a useful quantity that, to first order, is the time coordinate for final *stacked sections* and, to higher accuracy, is the time coordinate for *migrated time sections*. Even though v_{ins} is a general function of position, for the calculation of vertical traveltime, it can be considered as a function of depth alone because the calculation at each (x, y) is independent. Therefore, the instantaneous velocity will now be written $v_{\text{ins}}(z)$, with the understanding that the calculations can be done for any and all (x, y).

Vertical traveltime is calculated by considering a small depth interval, dz, over which v_{ins} can be considered to be approximately constant. The vertical traveltime over this interval is simply

$$d\tau = \frac{dz}{v_{\text{ins}}(z)}. \tag{6.2}$$

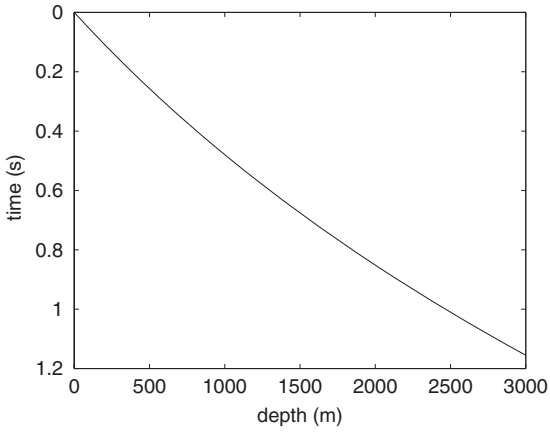
The total one-way traveltime from the surface to depth z is simply the sum of many such small contributions. In the limit as $dz \to 0$, this becomes the integral

$$\tau(z) = \int_0^z \frac{d\tilde{z}}{v_{\text{ins}}(\tilde{z})}. \tag{6.3}$$

The z dependence appears as the upper limit of the integral, while $\tilde{z}$ is just a dummy variable of integration. Defined in this way, $\tau(z)$ is a function that increases monotonically with z. As such, it is guaranteed to have an inverse. That is, given $\tau(z)$, it is always possible to find $z(\tau)$, and vice versa. The function $\tau(z)$ is called a *time–depth curve* and can be used to find the depth given the vertical traveltime or the reverse.

Continuing with the universal velocity function of Eq. (6.1), the vertical traveltime of Eq. (6.3) becomes

$$\tau(z) = \int_0^z \frac{d\tilde{z}}{v(\tilde{z})} = \frac{1}{c}\int_{v_0}^{v_0+cz} \frac{d\xi}{\xi} = \frac{1}{c}\ln\left[1 + \frac{cz}{v_0}\right]. \tag{6.4}$$



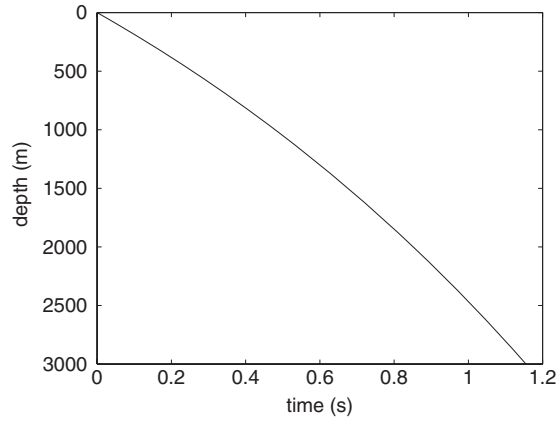


Figure 6.2a (left) The time–depth curve ($\tau(z)$) of Eq. (6.4) for the case of $v_0 = 1800$ m/s and $c = 0.6$/s.
Figure 6.2b (right) The depth–time curve ($z(\tau)$) of Eq. (6.4) for the case of $v_0 = 1800$ m/s and $c = 0.6$/s.

Thus a linear variation of velocity with depth yields a logarithmic time–depth curve. It is a simple matter to invert Eq. (6.4) for $z(\tau)$ to get

$$z(\tau) = \frac{v_0}{c}\left[e^{c\tau} - 1\right]. \tag{6.5}$$

For the case of the universal velocity function, the curves $\tau(z)$ and $z(\tau)$ are shown in Figures 6.2a and 6.2b, respectively. The graph of $z(\tau)$ is just the transpose of the graph of $\tau(z)$.

6.3 v_{ins} as a Function of Vertical Traveltime: $v_{\text{ins}}(\tau)$

In the preceding sections it has been shown that for any fixed (x, y) location, every depth z has a unique vertical traveltime associated with it. Therefore, v_{ins}, which is physically a function of z, may always be expressed as a function of $\tau(z)$. That is,

$$v_{\text{ins}}(\tau) = v_{\text{ins}}(z(\tau)). \tag{6.6}$$

Given $v_{\text{ins}}(\tau)$ in Eq. (6.2), an expression for $z(\tau)$ follows as

$$z(\tau) = \int_0^\tau v_{\text{ins}}(\tilde{\tau})\, d\tilde{\tau}. \tag{6.7}$$

Comparing Eqs. (6.3) and (6.7) shows that knowledge of $v_{\text{ins}}(z)$ allows computation of $\tau(z)$ and knowing $v_{\text{ins}}(\tau)$ enables computation of $z(\tau)$. In practice, $v_{\text{ins}}(z)$ might be directly obtained from a sonic well log, while $v_{\text{ins}}(\tau)$ can be estimated from an analysis of stacking velocities.

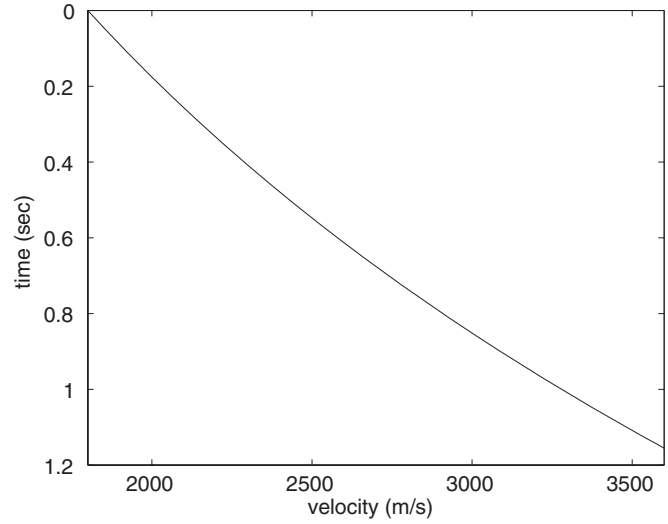


Figure 6.3 For the universal velocity function shown in Figure 6.1, v_{ins} is an exponential function of vertical traveltime.

For the universal velocity function of Eq. (6.1), the vertical traveltime results in the logarithmic expression given in Eq. (6.4) and $z(\tau)$ is given by Eq. (6.5). Thus $v_{ins}(\tau)$ becomes

$$v_{ins}(\tau) = v_{ins}(z(\tau)) = v_0 + c\left[\frac{v_0}{c}\left(e^{c\tau} - 1\right)\right], \tag{6.8}$$

which reduces to

$$v_{ins}(\tau) = v_0 e^{c\tau}. \tag{6.9}$$

Thus, when v_{ins} is linear with depth it is actually exponential with vertical traveltime. For the case of the universal velocity function, $v_{ins}(\tau)$ is plotted in Figure 6.3.

6.4 Average Velocity: v_{ave}

The typical industry definition of v_{ave} is that it is a particular depth divided by the vertical traveltime from the surface to that depth. Given any $z(\tau)$ curve, pick a point (τ_0, z_0) and v_{ave} is the slope of the line connecting the origin with the point (τ_0, z_0), while v_{ins} is the tangent to the curve at that point. Mathematically, $v_{ave}(z)$ is expressed as

$$v_{ave}(z) = \frac{z}{\tau(z)} = \frac{z}{\int_0^z d\tilde{z}/v_{ins}(\tilde{z})}, \tag{6.10}$$

while $v_{ave}(\tau)$ is

$$v_{ave}(\tau) = \frac{z(\tau)}{\tau} = \frac{1}{\tau}\int_0^\tau v_{ins}(\tilde{\tau})\, d\tilde{\tau}. \tag{6.11}$$

Equation (6.11) shows that the average velocity is a mathematical average only with respect to vertical traveltime. (Recall, from calculus, that the average (or mean) value of a

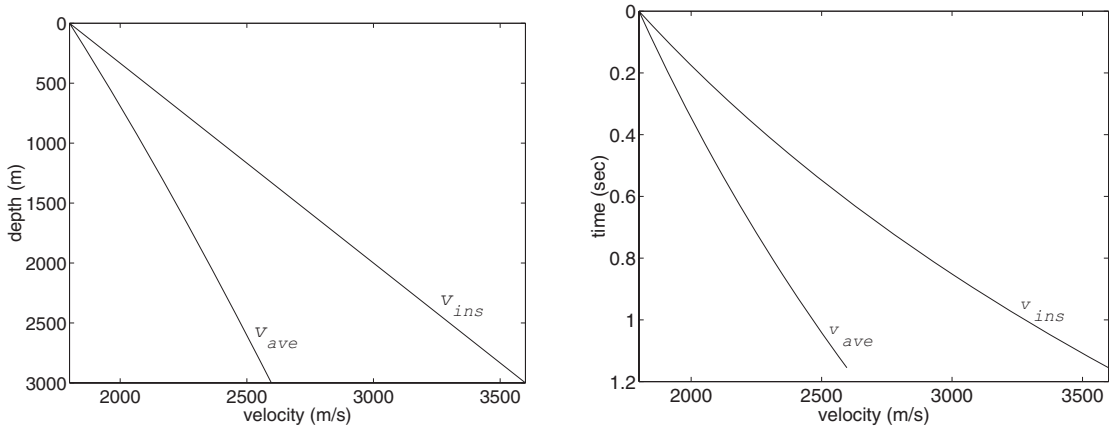


Figure 6.4a (left) v_{ave} and v_{ins} versus depth for the case of the universal velocity function (Eq. (6.1)) with $v_0 = 1800$ m/s and $c = 0.6\,\mathrm{s}^{-1}$.

Figure 6.4b (right) v_{ave} and v_{ins} versus time for the case of the universal velocity function (Eq. (6.1)) with $v_0 = 1800$ m/s and $c = 0.6\,\mathrm{s}^{-1}$.

function $f(x)$ over the interval from $0 \rightarrow x$ is $x^{-1}\int_0^x f(x')\,dx'$.) When considered as a function of depth, the average velocity is not an arithmetic average but in fact is mathematically the harmonic mean of velocities.

For the running example of a linear variation of velocity with depth, $v_{\mathrm{ave}}(z)$ becomes

$$v_{\mathrm{ave}}(z) = \frac{cz}{\ln\left[1 + cz/v_0\right]} \tag{6.12}$$

and $v_{\mathrm{ave}}(\tau)$ is then given by

$$v_{\mathrm{ave}}(\tau) = \frac{v_0}{c\tau}\left[e^{c\tau} - 1\right]. \tag{6.13}$$

These equations are graphed in Figures 6.4a and 6.4b.

6.5 Mean Velocity: v_{mean}

Since v_{ave} is a mathematical average of v_{ins} over traveltime and not depth, it is useful to define another velocity measure that is a depth average. The mean velocity, v_{mean}, is defined as

$$v_{\mathrm{mean}}(z) = \frac{1}{z}\int_0^z v_{\mathrm{ins}}(\tilde{z})\,d\tilde{z}. \tag{6.14}$$

For a linear variation of $v_{\mathrm{ins}}(z)$, $v_{\mathrm{mean}}(z)$ becomes

$$v_{\mathrm{mean}}(z) = \frac{1}{z}\int_0^z \left[v_0 + c\tilde{z}\right]\,d\tilde{z} = v_0 + \frac{1}{2}cz, \tag{6.15}$$

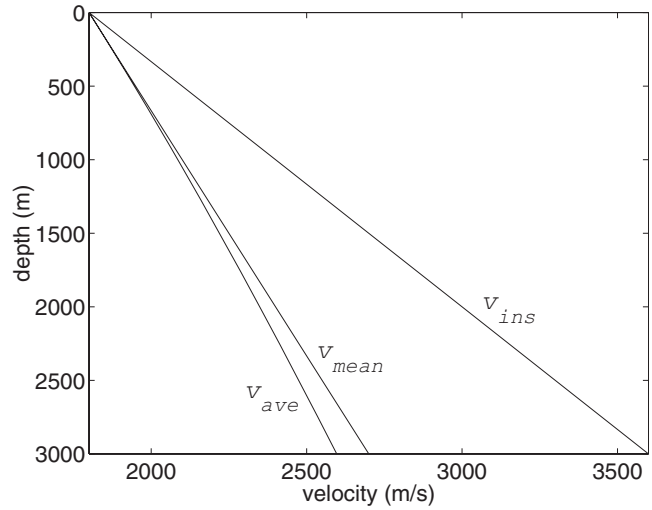


Figure 6.5 v_{mean} and v_{ave} are compared with v_{ins} for the case of a linear increase of v_{ins} with z.

which shows that the mean velocity increases at half the rate of $v_{ins}(z)$. Of course, $v_{mean}(z)$ can be reexpressed as a function of τ by substituting $z(\tau)$; this results in

$$v_{mean}(\tau) = \frac{v_0}{2}[1 + e^{c\tau}]. \tag{6.16}$$

Equation (6.15) is compared with $v_{ave}(z)$ and $v_{ins}(z)$ in Figure 6.5. $v_{mean}(z)$ and $v_{ins}(z)$ are both linear, but $v_{ave}(z)$ is not. Also, $v_{mean}(z)$ is always greater than $v_{ave}(z)$, which will be proven true in the next section.

6.6 RMS Velocity: v_{rms}

The mean velocity discussed in the previous section is not commonly used in seismology. The reason is that another velocity measure, v_{rms}, is used and only two of v_{ave}, v_{mean}, and v_{rms} are independent. The root mean square velocity is defined as

$$v_{rms}^2(\tau) = \frac{1}{\tau}\int_0^{\tau} v_{ins}^2(\tilde{\tau})\, d\tilde{\tau}. \tag{6.17}$$

Two important relationships exist between the three velocity measures v_{ave}, v_{mean}, and v_{rms}. First, they are linked by the relation

$$v_{rms}^2 = v_{ave} v_{mean}, \tag{6.18}$$

which means that only two of the three velocity measures can be considered independent. The proof is straightforward: $v_{rms}^2 = \tau^{-1}\int v_{ins}^2\, d\tau = (z/\tau)(1/z)\int v_{ins}[v_{ins}\, d\tau] = (z/\tau)[(1/z)\int v_{ins}\, dz] = v_{ave} v_{mean}$. This proof is simply a change of variables in the integration.

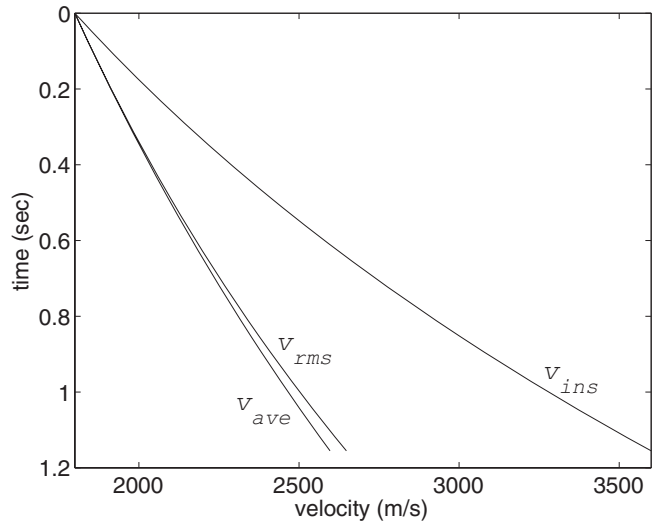


Figure 6.6 v_{rms} and v_{ave} are compared with v_{ins} for the case of a linear increase of v_{ins} with z.

It is also true that $v_{rms} \geq v_{ave}$. This second relationship follows from a mathematical inequality known as *Schwartz's inequality*. This result is quite general and confirms the intuition that the square root of the average of the squares of a set of numbers is always greater than the direct average of the numbers themselves. That is, let $\{x_k\}$ be an arbitrary set of n real numbers; then $\sqrt{n^{-1}\sum_k x_k^2} \geq n^{-1}\sum_k x_k$, where the equality occurs only if all x_k are identical.

For the continuing example of v_{ins} linear with depth, Eq. (6.9) showed that v_{int} was exponential. Thus v_{rms} becomes

$$v_{\text{rms}}^2(\tau) = \frac{1}{\tau}\int_0^\tau v_0^2 e^{2c\tilde{\tau}}\, d\tilde{\tau} = \frac{v_0^2}{2c\tau}\left[e^{2c\tau} - 1\right]. \tag{6.19}$$

Figure 6.6 compares v_{rms}, v_{ave}, and v_{ins} for the universal velocity function. Though v_{rms} is always greater than v_{ave}, the difference is not greater than 2%.

Exercises

6.6.1 Use MATLAB to numerically demonstrate Schwartz's inequality. Use `rand` to generate a vector of random numbers with zero mean. Compute both the mean and the rms average of these numbers. Repeat the process for many different vectors with steadily increasing length. Plot a graph of these two averages versus length. Repeat your work using random numbers that are all positive with a mean value somewhere near that expected for seismic velocities. Show that if the velocity vector has identical entries, then the rms average and the mean are equal but if one value differs from the rest, then the rms average exceeds the mean.

6.7 Interval Velocity: v_{ins}

Corresponding to any of the velocity averages, an *interval velocity* can be defined that is simply that particular average applied across a small interval rather than from the surface ($z = 0$) to depth z. For example, the average and rms velocities across an interval defined by τ_1 and τ_2 are simply

$$v_{\text{ave}}(\tau_2, \tau_1) = \frac{1}{\tau_2 - \tau_1} \int_{\tau_1}^{\tau_2} v_{\text{ins}}(\tilde{\tau})\, d\tilde{\tau} \tag{6.20}$$

and

$$v_{\text{rms}}^2(\tau_2, \tau_1) = \frac{1}{\tau_2 - \tau_1} \int_{\tau_1}^{\tau_2} v_{\text{ins}}^2(\tilde{\tau})\, d\tilde{\tau}. \tag{6.21}$$

These quantities are functions of both the upper and lower bounds of the integrals. In the limit, as the interval shrinks to be so small that $v_{\text{ins}}(\tau)$ can be considered constant, both interval velocities approach the instantaneous velocity.

It follows directly from the definition of average velocity (Eq. (6.10)) that the average velocity across a depth interval is just the ratio of the depth interval to the vertical traveltime across the interval. That is,

$$v_{\text{ave}}(\tau_2, \tau_1) = \frac{z_2 - z_1}{\tau_2 - \tau_1} = \frac{\Delta z}{\Delta \tau}. \tag{6.22}$$

Thus, if the range from 0 to z is divided into n finite intervals defined by $z_0, z_1, z_2, \ldots, z_{n-1}, z_n$ (where $z_0 = 0$ and $z_n = z$), then

$$v_{\text{ave}}(\tau) = \frac{z(\tau)}{\tau} = \frac{\sum_{k=1}^{n} [z_k - z_{k-1}]}{\tau} = \frac{1}{\tau} \sum_{k=1}^{n} [\tau_k - \tau_{k-1}] v_{\text{ave}}(\tau_k, \tau_{k-1}), \tag{6.23}$$

where $\tau_k = \tau(z_k)$. Defining $\Delta\tau_k = \tau_k - \tau_{k-1}$ then gives

$$v_{\text{ave}}(\tau) = \frac{1}{\tau} \sum_{k=1}^{n} v_k\, \Delta\tau_k, \tag{6.24}$$

where $v_k \equiv v_{\text{ave}}(\tau_k, \tau_{k-1})$ and $\tau = \sum_{k=1}^{n} \Delta\tau_k$. Comparing Eq. (6.24) with Eq. (6.11) suggests that the former is just a discrete version of the latter. However, the velocity v_k in Eq. (6.24) is the average velocity of the kth finite interval and is thus the time average of v_{ins} across the interval. Of course, if $v_{\text{ins}}(\tau)$ is constant across each interval, then v_k is an instantaneous velocity and this distinction vanishes.

Equation (6.24) can be used in a number of ways. Most obviously, it shows how to combine a set of *local average velocities* to obtain the *macro average velocity* across a larger interval. Also, it can be used to estimate a local average velocity given two macro average velocities. Suppose the average velocities v_{ave1} and v_{ave2} from $z = 0$ to depths z_1 and z_2 are

known. Then an expression for the average velocity from $z_1 \rightarrow z_2$ or, equivalently, from $\tau_1 \rightarrow \tau_2$ follows from Eq. (6.24) and is

$$v_{\text{ave}}(\tau_2, \tau_1) = \frac{1}{\tau_2 - \tau_1} \left[\tau_2 v_{\text{ave2}} - \tau_1 v_{\text{ave1}}\right]. \tag{6.25}$$

The two macro averages are each weighted by their time intervals and then the shallower average is subtracted from the deeper. This difference is then divided by the time interval of the local average.

If v_{ave1}, τ_1, v_{ave2}, and τ_2 are all measured without error, then this process (i.e., Eq. (6.25)) works perfectly. However, in a real case, errors in the measurements can lead to wildly incorrect estimates of $v_{\text{ave}}(\tau_2, \tau_1)$. With noisy data, there is no guarantee that the term $[\tau_2 v_{\text{ave2}} - \tau_1 v_{\text{ave1}}]$ will always be positive, leading to the possibility of negative velocity estimates. Also, the division by $\tau_2 - \tau_1$ can be unstable if the two times are very close together.

The discussion so far has been only for average velocities across an interval. The entire derivation above can be repeated for rms velocities with similar results, but a few more subtleties arise in interpretation. Rather than repeating the derivation just given with "rms" in place of "ave," it is instructive to consider an alternative approach. It is a property of integrals that $\int_a^c = \int_a^b + \int_b^c$, where $a < b < c$. Given $\tau_0 < \tau_1 < \tau_2$ and applying this rule to Eq. (6.17) results in

$$v_{\text{rms}}^2(\tau_2, \tau_0) = \frac{1}{\tau_2 - \tau_0} \left[\int_{\tau_0}^{\tau_1} v_{\text{ins}}^2(\tilde{\tau})\, d\tilde{\tau} + \int_{\tau_1}^{\tau_2} v_{\text{ins}}^2(\tilde{\tau})\, d\tilde{\tau}\right]. \tag{6.26}$$

Recognizing the integrals in $[\ldots]$ as rms interval velocities squared multiplied by their interval times (i.e., $\int_{\tau_0}^{\tau_1} v_{\text{ins}}^2(\tilde{\tau})\, d\tilde{\tau} = [\tau_1 - \tau_0] v_{\text{rms}}^2(\tau_1, \tau_0)$ and similarly for the other integral) leads to

$$v_{\text{rms}}^2(\tau_2, \tau_0) = \frac{1}{\tau_2 - \tau_0} \left[(\tau_1 - \tau_0) v_{\text{rms}}^2(\tau_1, \tau_0) + (\tau_2 - \tau_1) v_{\text{rms}}^2(\tau_2, \tau_1)\right]. \tag{6.27}$$

For the case of n subdivisions between τ_2 and τ_0, this generalizes to

$$v_{\text{rms}}^2(\tau_2, \tau_0) = \frac{1}{\tau_2 - \tau_0} \sum_{k=1}^{n} v_k^2 \Delta\tau_k, \tag{6.28}$$

where $v_k = v_{\text{rms}}(\tau_k, \tau_{k-1})$ and, as before, $\Delta\tau_k = \tau_k - \tau_{k-1}$ and $\tau = \sum_{k=1}^{n} \Delta\tau_k$. This is the rms equivalent of Eq. (6.24), and all of the comments made previously about v_{ave} apply here for v_{rms}. In particular, Eq. (6.28) should be thought of as combining interval rms velocities into a macro rms velocity. Only in the case when v_{ins} does not vary significantly across an interval can the v_k in Eq. (6.28) be considered to be instantaneous velocities.

Equation (6.27) is the *addition rule* for rms velocities. To combine rms velocities, the squared velocities are added and each must be weighted by its time interval. Equation (6.27) can be rearranged to give an expression for estimating a local rms velocity from two

macro velocities,

$$v_{rms}^2(\tau_2, \tau_1) = \frac{1}{\tau_2 - \tau_1}\Big[(\tau_2 - \tau_0)v_{rms}^2(\tau_2, \tau_0) - (\tau_1 - \tau_0)v_{rms}^2(\tau_1, \tau_0)\Big], \tag{6.29}$$

or, with simplified notation,

$$v_{rms}^2(\tau_2, \tau_1) = \frac{1}{\tau_2 - \tau_1}\Big[\tau_2 v_{rms2}^2 - \tau_1 v_{rms1}^2\Big]. \tag{6.30}$$

In this expression τ_0 has been set to 0 and $v_{rms2} = v_{rms}(\tau_2, \tau_0)$, and similarly for v_{rms1}. Equation (6.30) is often called the *Dix equation for interval rms velocities* because it was C. H. Dix (Dix, 1955) who first recognized the connection between stacking velocities and rms velocities and showed how to calculate an interval velocity from two stacking velocity measurements.

The application of Eq. (6.30) in practice requires the measurement of stacking velocities and vertical traveltimes to two closely spaced reflectors. Under the assumption that stacking velocities are well approximated by rms velocities (Dix, 1955; Taner and Koehler, 1969), the rms velocity of the interval is then estimated with Eq. (6.30). However, as with average velocities, errors in measurement lead to problems with the estimation of $v_{rms}(\tau_2, \tau_1)$. If the term $\left[\tau_2 v_{rms2}^2 - \tau_1 v_{rms1}^2\right]$ becomes negative, then imaginary interval velocity estimates result. Thus one essential condition for the use of this technique is that

$$v_{rms2}^2 > v_{rms1}^2 \frac{\tau_1}{\tau_2}. \tag{6.31}$$

Since $\tau_1/\tau_2 < 1$, this is a constraint upon how fast v_{rms} estimates can decrease with increasing time. There is no mathematical basis to constrain the rate at which v_{rms} can increase; however, it is reasonable to formulate a constraint on physical grounds. Since P-wave seismic velocities are not expected to exceed some v_{max} (say, 7000 m/s), a constraint would be

$$v_{rms2}^2 < v_{rms1}^2 \frac{\tau_1}{\tau_2} + v_{max}^2 \frac{\tau_2 - \tau_1}{\tau_2}. \tag{6.32}$$

Exercises

6.7.1 Show that the right-hand side of the inequality (6.32) is always greater than v_{rms1} provided that $v_{max} > v_{rms1}$, so that this is a constraint on the rate of increase of v_{rms}.

6.7.2 Suppose that the times and depths to two reflectors are known, say τ_1, τ_2, z_1, and z_2, and that the rms velocities to the reflectors are also known, say v_{rms1} and v_{rms2}. Consider the interval velocities defined by

$$v_{int} = \frac{z_2 - z_1}{\tau_2 - \tau_1} \quad \text{and} \quad \tilde{v}_{int} = \sqrt{\frac{v_{rms2}^2 \tau_2 - v_{rms1}^2 \tau_1}{\tau_2 - \tau_1}}.$$

Under what condition(s) will these interval velocity estimates be similar? If the interval between the reflectors is highly heterogeneous, which estimate will be larger? Why?

6.8 MATLAB Velocity Tools

Up to this point, the discussion of velocity measures has been illustrated with an instantaneous velocity whose form is a known analytic function. More realistically, velocity functions are inherently numerical because they are derived from experiment. The conversion of such numerical velocities from one form to another is often necessary in data processing, and software tools are required for this purpose. Two alternate approaches are (1) to fit the numerical velocity functions with an *n*th-order polynomial and perform the necessary integrations using polynomial integration rules, or (2) to implement a numerical integration scheme for the conversion formulas. The second approach is taken here.

Five functions are available to convert velocity functions from one form to another. There is also a utility plotting function to draw piecewise constant lines:

`vint2t` Computes vertical traveltime given interval velocity versus depth.
`vint2vave` Converts interval velocity to average velocity.
`vave2vint` Converts average velocity to interval velocity.
`vint2vrms` Converts interval velocity to rms velocity.
`vrms2vint` Converts rms velocity to interval velocity.
`drawvint` Plots interval velocity curves as piecewise constant functions.

No utility is provided to deal with instantaneous velocity, since this can be treated simply by generating a finely sampled interval velocity. Like seismic traces, velocity functions are specified by prescribing two vectors, one for the velocities and the other for the depth or time that they apply to. Interval velocities are inherently piecewise constant and the convention used here is that the kth velocity, v_k, prescribed at depth z_k, persists as a constant until depth z_{k+1}. Therefore, the last velocity in a velocity vector applies for all z greater than the last entry in the depth vector.

As a first example of the use of these functions, suppose that $v_{\text{ins}}(z)$ is a piecewise linear function with five "knees" prescribed by (v, z) pairs as (1200 m/s, 0 m), (2700 m/s, 500 m), (3500 m/s, 1400 m), (3000 m/s, 1600 m), (4000 m/s, 2000 m). Code Snippet 6.8.1 shows how to compute $v_{\text{ins}}(\tau)$ for this $v_{\text{ins}}(z)$ and displays its results in Figure 6.7a. Line 2 uses piecewise linear interpolation (`pwlint`) to resample the five-legged function to a fine interval so that the piecewise constant interval velocity approximates the desired piecewise linear instantaneous velocity. Line 5 computes $\tau(z)$ by calling `vint2t` and thus defining $v_{\text{ins}}(\tau)$. (Note that `vint2t` returns a *one-way time*, so these values must be doubled for the *two-way time*.)

The computation of $\tau(z)$ requires the numerical computation of the integral in Eq. (6.3). Given a densely sampled function, the simplest way to do a numerical integration is with

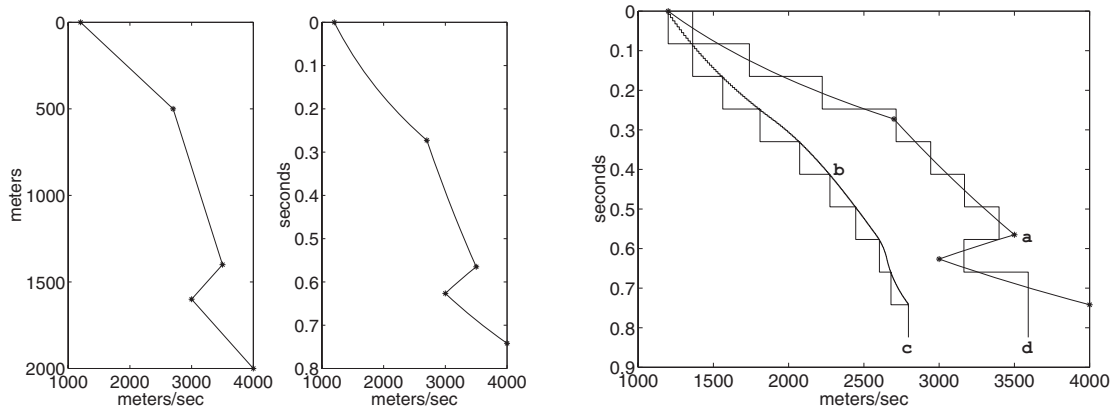


Figure 6.7a (left) (left panel) A five-legged $v_{\text{ins}}(z)$ with linear variation between the knees. (right panel) The curve on the left has been converted to $v_{\text{ins}}(\tau)$. The variation between the knees is no longer linear.

Figure 6.7b (right) (a) $v_{\text{ins}}(\tau)$ from Figure 6.7a. (b) Finely sampled $v_{\text{rms}}(\tau)$. (c) Desampled $v_{\text{rms}}(\tau)$. (d) Ten-legged interval rms approximation to $v_{\text{ins}}(\tau)$.

Code Snippet 6.8.1 This code defines a five-legged, piecewise linear $v_{\text{ins}}(z)$ and then computes $v_{\text{ins}}(\tau)$. It makes Figure 6.7a.

```
1  v=[1200 2700 3500 3000 4000];z=[0 500 1400 1600 2000];
2  z2=0:10:2000;v2=pwlint(z,v,z2);
3  subplot(1,2,1);plot(v2,z2,v,z,'r*');flipy
4  xlabel('meters/sec');ylabel('meters');
5  t2=vint2t(v2,z2);
6  t=interp1(z2,t2,z);
7  subplot(1,2,2);plot(v2,t2,v,t,'r*');flipy;
8  xlabel('meters/sec');ylabel('seconds');
```

End Code

velocitycode / velex1 . m

the functions `sum` and `cumsum`. The difference between these is that the first does a *definite integral*, while the second does an *indefinite integral*. For example, the command `sum([1:5].^2)` just adds the squares of the integers from 1 to 5 to get 55. This is a discrete approximation to the definite integral of the function $y = x^2$ from 0.5 to 5.5, for which the analytic answer is $(5.5^3 - 0.5^3)/3 \approx 55.4167$. Alternatively `cumsum([1:5].^2)` has the result $[1, 5, 14, 30, 55]$, which gives the value of the integral of $y = x^2$ for a lower limit of 0.5 and five successive upper limits of 1.5, 2.5, 3.5, 4.5, and 5.5 . Thus `sum` approximates $\int_{0.5}^{5.5} x^2\, dx$ and is just a single number, while `cumsum` approximates $\int_{0.5}^{x} \tilde{x}^2\, d\tilde{x}$ and results in a vector the same length as the input. Thus, the traveltime integration of Eq. (6.3) is done within *vint2t* with the single command `t1(2:nz)=cumsum(dz./vint(1:nz-1))`. The vector `t1` is $\tau(z)$ and has been initialized to zero, `dz` is a vector of the depth intervals, and `vint` is the interval velocity vector. (For more details, browse the source code file.)

Code Snippet 6.8.2 This carries on after Code Snippet 6.8.1 to compute v_{rms} from the surface for every point in $v_{\text{ins}}(\tau)$ a 10-point sparse $v_{\text{rms}}(\tau)$ and a 10-legged rms interval velocity approximation to $v_{\text{ins}}(\tau)$. Figure 6.7b is the result.

```
1  plot(v2,t2,v,t,'r*');flipy
2  vrms=vint2vrms(v2,t2);
3  tblock=linspace(min(t2),max(t2),10);
4  vrmsblock=vint2vrms(v2,t2,tblock);
5  drawvint(t2,vrms);drawvint(tblock,vrmsblock);
6  vint=vrms2vint(vrmsblock,tblock);
7  drawvint(tblock,vint);
8  xlabel('meters/sec');ylabel('seconds');
```

End Code

velocitycode / velex2 . m

Often velocity functions from well logs do not begin at $z = 0$. Therefore the computation of $\tau(z)$ from $z = 0$ requires that some initial velocity be used for the depth range above the log. This can be input to *vint2t*, in the form of a constant time shift τ_0, as the fourth argument. τ_0 defaults to $\tau_0 = z(1)/v(1)$, that is, the first depth divided by the first velocity. This can be estimated if the depth and time to some marker formation top are already known. Then *vint2t* can be run once with the initial velocity set to zero, and the time to the marker can be observed to be off by some $\Delta\tau$ in one-way time.

Now consider the computation of $v_{\text{rms}}(\tau)$ and the construction of a 10-layer approximation to $v_{\text{ins}}(\tau)$ using interval rms velocities. This is done in Code Snippet 6.8.2, and the result is shown in Figure 6.7b. Line 2 creates the dense $v_{\text{rms}}(\tau)$ function using *vint2vrms*, shown as curve "b" in the figure. Line 3 defines a blocky 10-legged time vector and line 4 creates the coarse $v_{\text{rms}}(\tau)$ (curve "c") by calling *vint2vrms* with a third argument. Essentially, this is just a desampled version of the finely sampled $v_{\text{rms}}(\tau)$. (It was not necessary to create the finely sampled $v_{\text{rms}}(\tau)$, as this was done to demonstrate the software.) Finally, line 6 creates the interval rms approximation to $v_{\text{ins}}(\tau)$ by sending the coarse $v_{\text{rms}}(\tau)$ to *vrms2vint*.

Exercises

6.8.1 Load the file `vp_from_well.mat`, which contains a vector of P-wave velocities and a corresponding vector of depths. Create two 10-leg interval-velocity approximations to $v_{\text{ins}}(z)$ one using average velocities and one using rms velocities. Compare the accuracy of time-to-depth conversion using these two alternate approximations. (For time-to-depth conversions, compute average velocities from both interval velocity functions.) Is there any significant difference?

6.8.2 Create a function called *vrms2vave* that converts average velocities to rms velocities. Then create the inverse function *vave2vrms*. Test your codes with the universal velocity function by comparing with the analytic forms for v_{rms} and v_{ave}.

6.8.3 Working with the universal velocity function, create $v_{\text{rms}}(\tau)$ for a depth range of 0–3000 m. Then use MATLAB's random number generator, `rand`, to add a 2% random fluctuation to the $v_{\text{rms}}(\tau)$ function. Finally, calculate $v_{\text{ins}}(\tau)$ from the perturbed $v_{\text{rms}}(\tau)$ function and determine the percentage error in the $v_{\text{ins}}(\tau)$ estimates caused by the 2% error in $v_{\text{rms}}(\tau)$. What does this say about the sensitivity of interval velocity calculations to noise?

6.9 Apparent Velocity: v_x, v_y, v_z

Consider a wavefront arriving at an array of detectors. For simplicity, we use only two dimensions, (x, z), and let the wavefront make an angle θ with respect to the horizontal (θ is called the *emergence angle* of the wave). The detectors are spaced at intervals of Δx at $z = 0$ (Figure 6.8a). If the medium immediately beneath the receivers has an acoustic velocity v_0, then the wavefront arrives at $x + \Delta x$ later than it does at x by the delay

$$\Delta t = \frac{\sin\theta}{v_0}\Delta x. \tag{6.33}$$

Thus it appears to move along the array at a speed

$$v_x \equiv \frac{\Delta x}{\Delta t} = \frac{v_0}{\sin\theta}. \tag{6.34}$$

The quantity v_x is called an *apparent velocity* because the wavefront appears to move at that speed, even though its true speed is v_0. Apparent velocities are one of the four fundamental observables in seismic data, the others being position, time, and amplitude. The apparent velocity is never less than the real velocity and can range up to *infinity*. That is, $v_0 \leq v_x \leq \infty$. Since infinities are cumbersome to deal with, it is common to work with the inverse of v_x, called the time dip, $\Delta t/\Delta x$, or horizontal slowness, s_x. Another common convention for horizontal slowness is to call it p, which signifies the *ray parameter*.

Now, we enlarge the thought experiment to include an array of receivers in a vertical borehole, spaced at Δz. Reasoning similar to that used before shows that the arrival at $z - \Delta z$ is delayed from that at z by

$$\Delta t = \frac{\cos\theta}{v_0}\Delta z. \tag{6.35}$$

Therefore, the vertical apparent velocity, v_z, is

$$v_z \equiv \frac{\Delta z}{\Delta t} = \frac{v_0}{\cos\theta}. \tag{6.36}$$

Using simple trigonometry, it results that

$$\frac{1}{v_0^2} = \frac{1}{v_x^2} + \frac{1}{v_z^2}. \tag{6.37}$$

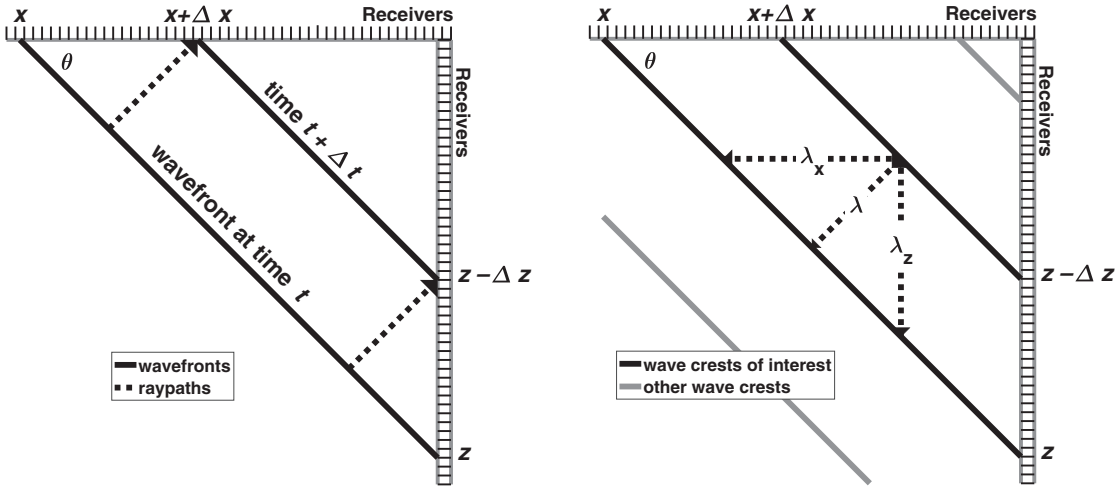


Figure 6.8a (left) An impulsive plane wave approaches horizontal and vertical receiver arrays.
Figure 6.8b (right) Similar to Figure 6.8a except that the plane wave is monochromatic.

If this argument is extended to 3D, the result is that there is an apparent velocity in the y direction as well and that the sum of the inverse squares of the apparent velocities is the inverse square of the real velocity:

$$\frac{1}{v_0^2} = \frac{1}{v_x^2} + \frac{1}{v_y^2} + \frac{1}{v_z^2}. \tag{6.38}$$

We observe apparent velocity, or equivalently time dip, by simply choosing an event of interest on a seismic record and measuring the slope, $\Delta t/\Delta x$. It will be seen shortly that this is a measurement of the ray parameter of the ray associated with the chosen event. Knowledge of the ray parameter essentially determines the raypath, provided that the velocities in the subsurface are known. This allows the ray to be projected down into the Earth and possibly allows us determine its reflection point. This technique, called *raytrace migration*, will be discussed in Section 7.2.2.

Another way to measure apparent velocities is with a multidimensional Fourier transform. In two dimensions, for example, the *f–k* transform represents a seismic record on a grid of horizontal wavenumber k_x and temporal frequency f. Each point in the (k_x,f) plane has a complex number associated with it that gives the amplitude and phase of a fundamental Fourier component, $e^{2\pi i(k_x x - ft)}$. In 3D, these fundamental components are monochromatic (i.e., with a single f) plane waves, so in 3D or 2D they are called *Fourier plane waves*. These waves are infinite in extent, so they have no specific arrival time. However, it does make sense to ask when a particular wave crest arrives at a particular location. Mathematically, a wave crest is identified as a point of constant phase, where *phase* refers to the entire argument of the complex exponential. If the Fourier transform has the value $Ae^{i\phi}$ at some (k_x,f), then the Fourier plane wave at that point is $Ae^{2\pi i(k_x x - ft)+i\phi}$. The motion of a point of constant phase is tracked by equating the phase at (x, t) with that

at $(x + \Delta x, t + \Delta t)$ as in

$$2\pi i(k_x x - ft) + i\phi = 2\pi i(k_x(x + \Delta x) - f(t + \Delta t)) + i\phi, \tag{6.39}$$

from which it follows that

$$\frac{\Delta x}{\Delta t} = \frac{f}{k_x}. \tag{6.40}$$

Thus the ratio of f to k_x determines the horizontal apparent velocity of the Fourier plane wave at (k_x,f). Radial lines (from the origin) in the (k_x,f) plane connect points of common apparent velocity. The advantage of the Fourier domain lies in this simultaneous measurement of apparent velocity for the entire seismic section. The disadvantage is that the notion of the spatial position of a particular apparent velocity measurement is lost.

If Eq. (6.38) is evaluated for the Fourier plane wave $e^{2\pi i(k_x x + k_y y + k_z z - ft)}$, the result is

$$\frac{1}{v^2} = \frac{k_x^2}{f^2} + \frac{k_y^2}{f^2} + \frac{k_z^2}{f^2}. \tag{6.41}$$

Now, for any monochromatic wave, we have $\lambda f = v$ and, using $k = \lambda^{-1}$, Eq. (6.41) leads to

$$k^2 = k_x^2 + k_y^2 + k_z^2. \tag{6.42}$$

This very important result shows that the coordinate wavenumbers, (k_x,k_y,k_z), are the components of a wavenumber vector, $\vec{k}$. Using a position vector $\vec{r} = (x, y, z)$, the Fourier plane wave can be written compactly as $e^{2\pi i(\vec{k}\cdot\vec{r} - ft)}$. Equation (6.42) can be expressed in terms of apparent wavelengths (e.g., $k_x = \lambda_x^{-1}$, etc.) as

$$\frac{1}{\lambda^2} = \frac{1}{\lambda_x^2} + \frac{1}{\lambda_y^2} + \frac{1}{\lambda_z^2}, \tag{6.43}$$

which shows that, like apparent velocities, the apparent wavelengths are always greater than the true wavelength. An important property of the wavenumber vector is that it points in the direction of wave propagation (for an isotropic, homogeneous medium). An easy way to see this is to take the gradient of the phase of a Fourier plane wave. Since a wavefront is defined as a surface of constant phase, wavefronts can be visualized as contours of the phase function $\tilde{\phi} = \pi i(\vec{k} \cdot \vec{r} - ft)$. The gradient of this phase function points in the direction normal to the wavefronts, that is, the direction of a raypath. This gradient is

$$\vec{\nabla}\tilde{\phi} = \frac{\partial\tilde{\phi}}{\partial x}\hat{x} + \frac{\partial\tilde{\phi}}{\partial y}\hat{y} + \frac{\partial\tilde{\phi}}{\partial z}\hat{z}, \tag{6.44}$$

where $\hat{x}$, $\hat{y}$, and $\hat{z}$ are unit vectors in the coordinate directions. Calculating the indicated partial derivatives leads to

$$\vec{\nabla}\tilde{\phi} = k_x\hat{x} + k_y\hat{y} + k_z\hat{z} = \vec{k}. \tag{6.45}$$

This equation can be achieved more simply by writing $\vec{\nabla} = \hat{r}\,\partial/\partial r$, where $r = \sqrt{x^2 + y^2 + z^2}$ and $\hat{r}$ is a unit vector pointing to a particular location, so that

$$\vec{\nabla}\tilde{\phi} = \hat{r}\frac{\partial \vec{k} \cdot \vec{r}}{\partial r} = \hat{r}\left|\vec{k}\right| = \vec{k}. \tag{6.46}$$

So, the inverse apparent velocities are proportional to the components of the wavenumber vector that points in the direction of wave propagation.

6.10 Snell's Law

Snell's law is the relation that governs the angles of reflection and refraction of wavefronts (or equivalent raypaths) at velocity interfaces. To understand its origins, consider Figure 6.9, which shows a periodic plane wave propagating across a planar interface between two media of velocities v_1 and v_2. In the first medium, the wavelength is $\lambda_1 = v_1/f$, while in the second medium it is $\lambda_2 = v_2/f$. As the wavefronts cross the interface, they must do so in a continuous fashion, otherwise, the wavefronts in the second medium would either lead or lag those in the first medium. Either possibility violates considerations of causality. The only way for the two wavefront systems to maintain continuity across the interface, and still have different wavelengths, is for them to have different angles with respect to the interface. The relationship between these angles follows from a consideration of apparent velocity. Suppose an array of receivers were placed along the interface. Then the requirement of wavefront continuity means that the wavelength component *measured along the interface* must be the same when evaluated in either medium. Since the frequency f does not change (this is a property of linear wave propagation), the apparent velocities measured along the interface must be the same in both media. Working in the rotated coordinates (x', z'), the apparent velocity $v_{x'}$ must give the same result when

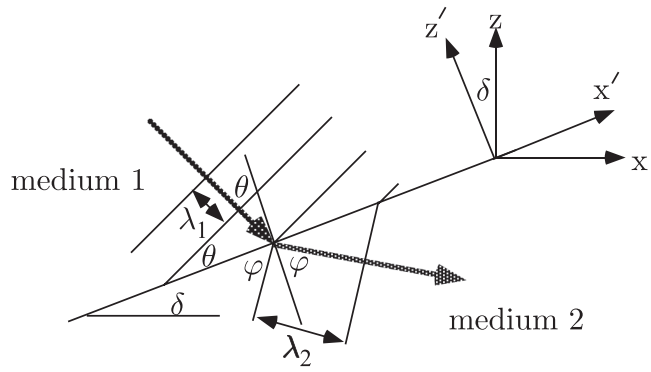


Figure 6.9 Snell's law results from the physical requirement that the apparent velocity along an interface be conserved.

evaluated using v_1 and θ as when using v_2 and ϕ. Thus

$$v_{1x'} \equiv \frac{v_1}{\sin\theta} = v_{2x'} \equiv \frac{v_2}{\sin\phi}. \tag{6.47}$$

This result is called *Snell's law*. The angles involved in Snell's law can be considered as the angles between the wavefronts and the interface or between the raypaths and the normal to the interface.

Snell's law can be derived in other ways, perhaps the most common being to demonstrate that it is a consequence of requiring the raypaths correspond to *stationary traveltimes*. This is known as *Fermat's principle*. Physically, waves can propagate energy from point A to point B along infinitely many different paths. Fermat's principle says that the most important paths are those for which the traveltime is stationary with respect to slight variations of the path. Here, *stationary* most often means a *minimum*, though there are important physical situations for which traveltime is maximized.

The derivation of Snell's law given here makes it clear that it applies to reflected as well as transmitted energy. In an acoustic medium, where there is only one mode of wave propagation, this results in the angle of incidence and the angle of reflection being equal. However, in elastic media, the incident and reflected waves can be either P-waves or S-waves. For example, in the case of an incident P-wave reflecting as an S-wave, Snell's law states that

$$\frac{\sin\theta_P}{v_P} = \frac{\sin\theta_S}{v_S}. \tag{6.48}$$

As a final summary statement, Snell's law states that wavefront propagation across a velocity interface always conserves the apparent velocity along the interface. Though it is not proven here, this holds for arbitrary wave types and for nonplanar wavefronts and interfaces.

6.11 Ray Tracing in a $v(z)$ Medium

A $v(z)$ medium is one where $\partial_x v = \partial_y v = 0$ and all velocity interfaces are horizontal. In this case, Snell's law says that the horizontal apparent velocity of the wavefront associated with the ray is conserved. Using the notation of Figure 6.10, this may be stated, for the jth interface, as

$$\frac{\sin\theta_{j-1}}{v_{j-1}} = \frac{\sin\phi_j}{v_j}. \tag{6.49}$$

Since all of the velocity interfaces are horizontal, $\phi_j = \theta_j$, so that

$$\frac{\sin\theta_{j-1}}{v_{j-1}} = \frac{\sin\theta_j}{v_j}. \tag{6.50}$$

This analysis may be repeated for any other interface with a similar conclusion. Thus the quantity $p = \sin\theta_j / v_j$ is conserved throughout the entire wave propagation. p is generally referred to as the *ray parameter*, since it is a unique constant for any ray and so

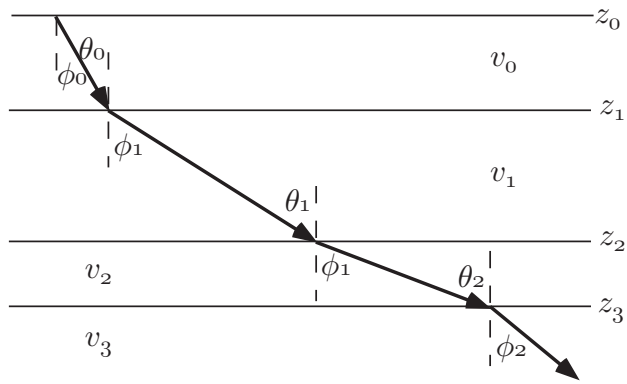


Figure 6.10 In a $v(z)$ medium, all velocity interfaces are horizontal and ray propagation is especially simple.

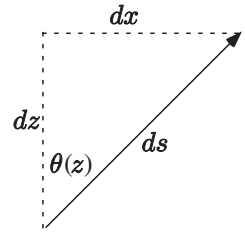


Figure 6.11 A differential ray element *ds*, at depth *z*, traveling at an angle $\theta(z)$ with respect to the vertical.

parameterizes (or names) the possible rays. The conservation of p and its identification as the horizontal apparent slowness is a direct consequence of Snell's law. This analysis generalizes to a continuous variation of v with z such that

$$p \equiv \frac{\sin(\theta(z))}{v(z)} = \text{a constant for any particular ray.} \tag{6.51}$$

General expressions for the traveltime and the horizontal distance traveled by any ray in a $v(z)$ medium can be easily derived. Figure 6.11 shows a differential element of a ray. From the geometry, it follows that

$$dx = \tan(\theta(z))\ dz \tag{6.52}$$

and

$$dt = \frac{ds}{v(z)} = \frac{dz}{v(z)\cos(\theta(z))}. \tag{6.53}$$

Snell's law is incorporated by replacing the trigonometric functions using $pv(z) = \sin(\theta(z))$ so that

$$dx = \frac{pv(z)}{\sqrt{1 - p^2 v^2(z)}}\, dz \tag{6.54}$$

and

$$dt = \frac{dz}{v(z)\sqrt{1 - p^2 v^2(z)}}. \tag{6.55}$$

Expressions for macroscopic raypaths are obtained by simply integrating these results. For a ray traveling between depths z_1 and z_2, the horizontal distance traveled becomes

$$x(p) = \int_{z_1}^{z_2} \frac{pv(z)}{\sqrt{1 - p^2 v^2(z)}}\, dz \tag{6.56}$$

and the total traveltime is

$$t(p) = \int_{z_1}^{z_2} \frac{dz}{v(z)\sqrt{1 - p^2 v^2(z)}}. \tag{6.57}$$

Given a velocity function and a ray parameter, we can compute exactly the horizontal distance (offset) and traveltime for a ray that traverses between two depths. The difficulty is that it is usually desired to trace a ray with a specific offset, and there is no simple way to determine the ray parameter that will do this. Generally the process is iterative. The offsets produced by a fan of rays (i.e., p values) are examined and, hopefully, two p values will be found that bracket the desired offset. Then a new, refined fan of rays can be constructed and the process repeated until a ray is found that produces the desired offset within a specified tolerance (called the capture radius).

If the $v(z)$ medium is discretely layered rather than continuously variable, then summation forms of Eqs. (6.56) and (6.57) are more appropriate. These are easily seen to be

$$x(p) = \sum_{k=1}^{n} \frac{pv_k}{\sqrt{1 - p^2 v_k^2}}\, \Delta z_k \tag{6.58}$$

and

$$t(p) = \sum_{k=1}^{n} \frac{\Delta z_k}{v_k \sqrt{1 - p^2 v_k^2}}. \tag{6.59}$$

6.11.1 Measurement of the Ray Parameter

Given a seismic source record, the ray parameters for the upcoming waves arriving at the receiver spread may be estimated by measuring the horizontal apparent velocity of each wave (see Section 6.9). As shown in Figure 6.12, the emergence angle θ_0 of a ray arriving at a particular pair of receivers is given by

$$\sin\theta_0 = \frac{v_0\, \Delta t}{\Delta r}, \tag{6.60}$$

where v_0 is the instantaneous velocity immediately beneath the receivers, Δr is the receiver spacing, and Δt is the time delay between the wavefront arrivals at r and $r + \Delta r$. In arriving at this expression, any wavefront curvature has been assumed to be inconsequential

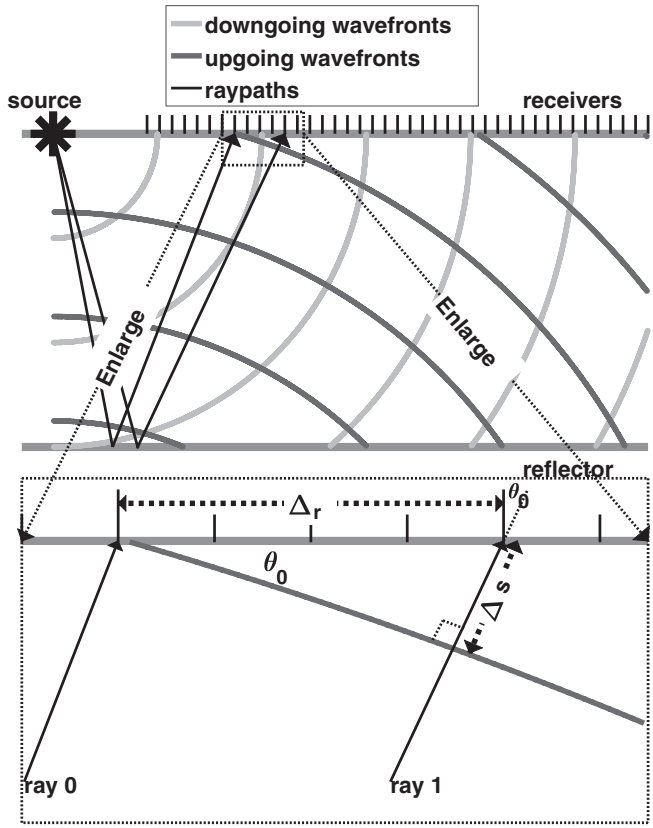


Figure 6.12 The ray parameter can be measured directly by measuring the delay between wavefront arrivals at two closely spaced receivers. The upper portion of this figure shows the wavefronts and raypaths for a reflection from a single horizontal reflector. The lower portion is an enlargement of the small dotted rectangle in the upper part (omitting the downgoing wavefront) and shows the geometry needed to estimate the ray parameter of the reflected wave at a particular point. "Ray 0" emerges at receiver r_0 and "ray 1" at receiver r_1, where $r_1 = r_0 + \Delta r$. The wavefront makes an *emergence angle* θ_0 with the receiver plane. The traveltime measured at r_1 is greater than that at r_0 owing to the extra path length Δs. The two receivers must be chosen sufficiently close such that wavefront curvature is negligible between them.

at the local site of measurement. According to Eq. (6.51), the ray parameter is given by $\sin\theta_0/v_0$, so

$$\frac{\Delta t}{\Delta r} = \frac{1}{v_r} = \frac{\sin\theta_0}{v_0} = p, \tag{6.61}$$

where v_r is the horizontal apparent velocity of the wave at the measurement location.

Generally, it is expected that p will change with position owing to changes in emergence angle and to lateral variations in the instantaneous velocity. Horizontal events, i.e., waves traveling vertically when they reach the receiver array, have a ray parameter of 0. Waves traveling horizontally across the array have a ray parameter of $1/v_0$ and on a seismic (x, t) plot have the maximum possible slope. Even though the wavefronts are vertical, the slope

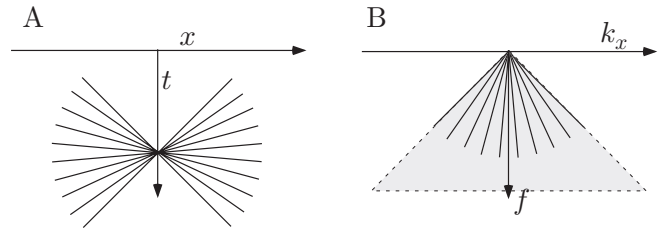


Figure 6.13 The physical considerations that place limits upon the ray parameter manifest as limits on the maximum time dip in (x, t) space (A) and segment f–k space into allowed and forbidden regions (B).

on the time section cannot exceed $1/v_0$. Taking sign into account, the range of possible values for the ray parameter is $-v_0^{-1} \leq p \leq v_0^{-1}$. This maximum possible steepness in (x, t) space is a fundamental property of a seismic time section. Assuming a value for v_0 in the course of data processing means that any events with slopes steeper than v_0^{-1} are interpreted as nonphysical and must be rejected. Since apparent velocities can also be expressed in f–k space as f/k_x (Eq. (6.40)), this phenomenon means that f–k space is segmented into allowed and forbidden regions.

Figure 6.13 shows this situation for both (x, t) and f–k space. In (x, t) space, the fan of allowed time dips forms a bow tie shape when plotted at a specific point, though such events may exist at all (x, t) locations. In f–k space, the zero ray parameter plots vertically down the f axis, while it is horizontal in (x, t) space. Thus the fan of allowed p values in f–k space forms a symmetric shape about the f axis. (Only one half of the bow tie is shown here, because negative frequencies are generally redundant for real-valued data.) In f–k space, p values are found only along radial lines emanating from the origin, not along lines of the same slope emanating from any other point. The portion of f–k space corresponding to $|k_x/f| < v_0^{-1}$ (i.e., outside the shaded region in Figure 6.13B) is "forbidden" to simple rays and is called the *evanescent* region. (It will be seen later that certain exponentially decaying wave types can occur here.) The shaded portion of Figure 6.13B is called the *body wave region* and is the portion of f–k space that is available for seismic imaging.

6.11.2 Raypaths when $v = v_0 + cz$

It is not difficult to integrate Eqs. (6.56) and (6.57) for the case of an instantaneous velocity that increases linearly with depth (e.g., the universal velocity function). The details can be found in Slotnick (1959), pp. 205–211. Letting $v(z) = v_0 + cz$, $z_1 = 0$, and $z_2 = z$, the results are

$$x(z,p) = \frac{1}{pc}\left[\sqrt{1 - p^2 v_0^2} + \sqrt{1 - p^2 \{v_0 + cz\}^2}\right] \tag{6.62}$$

and

$$t(z,p) = \frac{1}{c} \ln\left(\left[\frac{v_0 + cz}{v_0}\right]\left[\frac{1 + \sqrt{1 - p^2 v_0^2}}{1 + \sqrt{1 - p^2 \{v_0 + cz\}^2}}\right]\right). \tag{6.63}$$

Slotnick shows that Eq. (6.62) describes an arc of a circle, having radius $1/(pc)$ and centered at $x_0 = \sqrt{1 - p^2 v_0^2}/(pc)$ and $z_0 = -v_0/c$. Manifestly, Eq. (6.62) describes a ray as it goes from zero to some depth z. Since the velocity is always increasing in this case, the Snell's law angles always become larger as the ray goes deeper. Eventually the ray flattens out, when $\theta(z) = \sin^{-1}(pv(z)) = 90°$, and turns upward. Therefore, the depth at which a ray bottoms out, called its *turning point*, is found from

$$z_{\max} = \frac{1 - pv_0}{pc}. \tag{6.64}$$

The complete raypath for a ray that reaches its turning point and then returns to the surface must have two values of x for each z, so the function $x(z)$ is mathematically double valued. However, $z(x)$ is still single valued, so the complete raypath is most easily computed by solving Eq. (6.62) for z to get

$$z = \frac{1}{pc}\left[\sqrt{1 - \{pcx - \cos\theta_0\}^2} - pv_0\right]. \tag{6.65}$$

Code Snippet 6.11.1 implements Eq. (6.65) to create the raypath display shown in Figure 6.14a. This code establishes a horizontal distance range and a set of takeoff angles in lines 1–3. Then, in looping over the desired rays, it calculates a depth for each element of the vector `x`. However, many of these rays will not cover the full range of `x`. Equation (6.65) returns a complex number for a distance `x` that cannot be reached by a particular ray. Lines 11–14 search for these points and replace their depths with `NaN`. (Recall that `NaN`'s do not display when plotted.) Also, the code generates z values that are negative, so lines 15–18 set these to `NaN`. The ray bending in Figure 6.14a is much more realistic than a simple constant-velocity calculation using straight rays. It shows that most of the energy of a seismic source cannot penetrate to great depths, because it is turned around by refraction. All of the raypaths are obviously circular arcs whose centers move to the right as the ray parameter decreases.

Slotnick also derives expressions for the wavefronts (surfaces of constant traveltime) and shows them to be circles whose centers are along the z axis at

$$z_{w0} = \frac{v_0}{c}\left[\cosh(cT) - 1\right], \tag{6.66}$$

where T is the traveltime defining the wavefront. Each circular wavefront has a radius of

$$r = \frac{v_0}{c}\sinh(cT). \tag{6.67}$$

Code Snippet 6.11.2 calculates and plots 10 wavefronts for a set of traveltimes from 0 to 5 s. For each wavefront, the strategy is to calculate the center of the wavefront circle (line 7) and its radius (line 8). Then, for a predefined set of depths (line 1), the horizontal positions are calculated from the equation of a circle (line 9). As in the case of the raypaths, this results in both complex and negative values, which are found and set to `NaN` (lines 10–17). The resulting wavefronts clearly show the effects of increasing velocity with

Code Snippet 6.11.1 This code implements Eq. (6.65) and makes Figure 6.14a.

```
x=1:50:35000;
vo=1800;c=.6;nrays=10;
thetamin=5;thetamax=80;
deltheta=(thetamax-thetamin)/nrays;
zraypath=zeros(nrays,length(x));
for k=1:nrays
        theta=thetamin+(k-1)*deltheta;
        p=sin(pi*theta/180)/vo;
        cs=cos(pi*theta/180);
        z = (sqrt( 1/(p^2*c^2) - (x-cs/(p*c)).^2) -vo/c);
        ind=find(imag(z)~=0.0);
        if(~isempty(ind))
                z(ind)=nan*ones(size(ind));
        end
        ind=find(real(z)<0.);
        if(~isempty(ind))
                z(ind)=nan*ones(size(ind));
        end
        zraypath(k,:) = real(z);
end
figure;plot(x/1000,zraypath/1000);flipy;
xlabel('x kilometers');ylabel('z kilometers')
```

End Code

velocitycode / slotnick1 .m

depth, being more closely spaced at shallow depths than when deeper. Figure 6.15 superimposes the raypaths and wavefronts and uses the command `axis equal` to ensure a 1 : 1 aspect ratio. Clearly, the raypaths are normal to the wavefronts.

Exercises

6.11.1 Use Eqs. (6.62) and (6.63) to calculate and plot the (x, t) curve for a P–P reflection from 2000 m depth. Assume the universal velocity function and a source and receivers at depth 0. Repeat your calculations for a P–S reflection assuming a constant v_P/v_S of 2. In each case, what is the maximum source–receiver offset for which a reflection is expected?

6.11.2 Derive Eqs. (6.62) and (6.63) from Eqs. (6.56) and (6.57).

6.11.3 For a linear increase of velocity with depth, a source at $(x, z) = (0, 0)$, and considering transmitted rays only, is there a unique p for each point in the (x, z) plane? Will your conclusion remain valid for a more general $v(z)$?

6.11.3 MATLAB Tools for General $v(z)$ Ray Tracing

The analytic expressions in the previous section produce first-order realism by including the effects of ray bending when $v(z)$ is a linear function of depth. However, more accurate

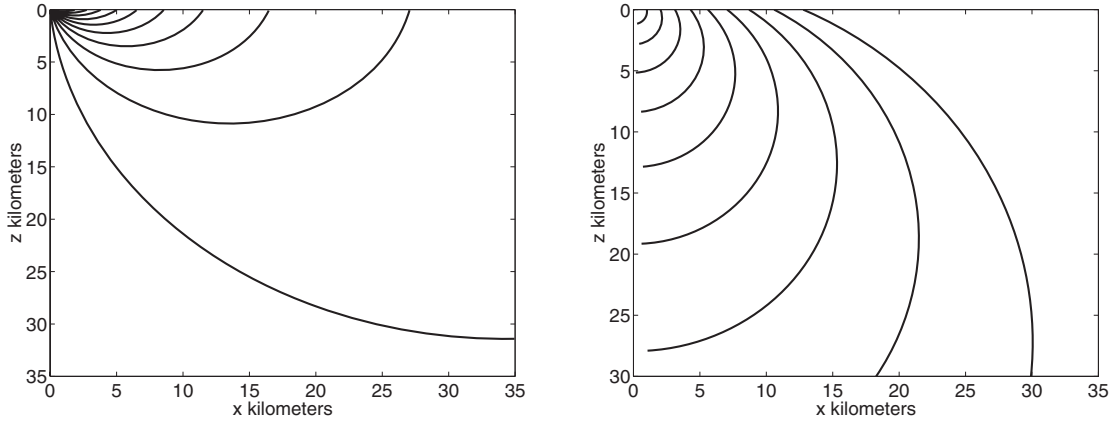


Figure 6.14a (left) A selection of raypaths are shown for the universal velocity function of Figure 6.1. Code Snippet 6.11.1 created this plot.

Figure 6.14b (right) A set of wavefronts associated with the raypaths of Figure 6.14a. This was created with Code Snippet 6.11.2.

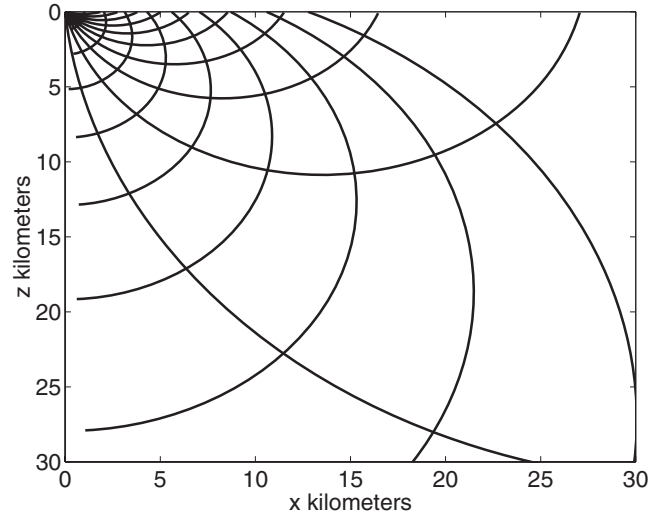


Figure 6.15 This figure superimposes the raypaths of Figure 6.14a onto the wavefronts of Figure 6.14b.

results are often desired for complicated $v(z)$'s such as those that result from well-log measurements. In this case the only way to proceed is through a numerical implementation of Eqs. (6.56) and (6.57), or more properly, Eqs. (6.58) and (6.59). Usually, it is desired to trace rays between two specific points such as a source and a receiver, perhaps via a reflection at a specific depth. There is no known solution technique that solves this *two-point ray tracing* problem in one step. Instead, an iterative procedure, as alluded to on page 329, must be used.

For definiteness, suppose it is desired to raytrace a P–P reflection from a reflector at $z_r = 2000\,\text{m}$ in an arbitrary $v(z)$ medium and for offsets from 100 to 2000 m. Since a P–P

Code Snippet 6.11.2 This code assumes that Code Snippet 6.11.1 has already been run and proceeds from there to calculate and plot the wavefronts. Figure 6.14b is the result.

```
zw=0:50:30000;
nwaves=10;tmax=5;
xwavefront=zeros(nwaves,length(zw));
times=linspace(0,tmax,nwaves);
zo=zeros(1,nwaves);
for k=1:nwaves
   zo(k)=vo*(cosh(c*times(k))-1)/c;
   r=vo*sinh(c*times(k))/c;%radius
   xw=sqrt(r.^2-(zw-zo(k)).^2);
   ind=find(real(xw)<0.);
   if(~isempty(ind))
      xw(ind)=nan*ones(size(ind));
   end
   ind=find(imag(xw)~=0.0);
   if(~isempty(ind))
      xw(ind)=nan*ones(size(ind));
   end
   xwavefront(k,:) = real(xw);
end
figure;plot(xwavefront/1000,zw/1000);flipy;
xlabel('x kilometers');ylabel('z kilometers')
```

End Code

velocitycode / slotnick2 .m

ray follows a completely symmetric path (if source and receiver are at the same depth), it is sufficient to trace a ray from $z = 0$ to $z = z_r$ at offset $x/2$ and then double the results. However, Eq. (6.56) does not easily facilitate this, since the value of p that will do the job is not known. If the velocity were constant, then simple geometry would predict a takeoff angle of $\theta_0 = \arctan(x/(2z_r))$ and thus the ray parameter would be trivially found. For increasing velocity with depth, the takeoff angle will be smaller than that predicted with constant velocity, while for decreasing velocity with depth the situation is reversed. In the general case, the takeoff angle and hence the ray parameter cannot be found analytically.

There are seven functions and one demonstration script provided for $v(z)$ ray tracing. These are:

rayfan Shoots a fan of rays given their ray parameters for $v(z)$.

rayfan_a Similar to *rayfan* but the rays are specified by angle.

shootray Similar to *rayfan* but with less error checking (faster).

traceray_pp Traces a P–P (or S–S) reflection for $v(z)$.

traceray_ps Traces a P–S (or S–P) reflection for $v(z)$.

traceray Traces an arbitrary ray given its ray code for $v(z)$.

drawray Plots rays given their ray parameters.

raytrace_demo Interactive demonstration of $v(z)$ ray-tracing capabilities.

The approach taken here is to shoot a *fan* of rays and to iteratively refine the fan until a convergence criterion is met. For a general case, this iteration will shoot many fans of rays, so the ray fan algorithm needs to be very efficient. The functions *rayfan*, *rayfan_a*, and *shootray* all perform this task in a similar fashion. Only the latter is actually used in the two-point ray tracers *traceray_pp*, *traceray_ps*, and *traceray*, because it is the most efficient. However, the efficiency of *shootray* is achieved with the sacrifice of some flexibility, so *rayfan* and *rayfan_a* are provided as well.

As shown in Code Snippet 6.11.3, *shootray* has three inputs, `v`, `z`, and `p`, which are respectively column vectors of velocity and depth and a row vector of ray parameters. The vector of rays defined by `p` is traced from `z(1)` to `z(end)` using the velocities in `v`. As discussed in Section 6.8, the velocity model is assumed to be piecewise constant, with velocity `v(k)` beginning at `z(k)` and persisting as a constant until `z(k+1)`. Accordingly, the last velocity in `v` is irrelevant for *shootray*, and line 3 establishes an index vector to the relevant velocities. Lines 4 through 8 are key to the efficiency of *shootray*. On line 4, `v(iprop)` is a column vector of length m and `p` is a row vector of length n. This product of an $m \times 1$ matrix representing $v(z)$ with a $1 \times n$ matrix representing p is an $m \times n$ matrix of $\sin\theta = pv(z)$ called `sn`. The kth column of `sn` represents the product $pv(z)$ for the kth ray parameter. Line 6 builds the matrix `cs` that represents $\cos\theta(z) = \sqrt{1 - p^2 v^2(z)}$. Lines 7 and 8 build the $m \times n$ matrices `vprop` and `thk` that represent $v(z)$ and Δz for each ray. Since these quantities are the same for each ray, they are represented by matrices with identical columns. Thus, we have four $m \times n$ matrices, with z as the row coordinate and p as the column coordinate. Not all of these combinations of z and p can correspond to physical rays, since the depth of penetration of a ray is limited. Line 5 detects these nonphysical combinations by finding any values of $pv(z)$ that are greater than one.

The ray tracing is completed in lines 9–15 and an `if` statement is used to handle the single-layer case. Using the $m \times n$ matrices `thk`, `sn`, and `cs`, line 10 computes Eq. (6.58) by using the `.*` operator to form the expression $pv_k \, \Delta z_k / \sqrt{1 - p^2 v_k^2}$ as an $m \times n$ matrix. Then, as discussed in Section 6.8, the function `sum` is used to compute the discrete sum that approximates the integral in Eq. (6.56). When applied to a matrix, `sum` produces a *row* vector that is the sum of each *column* of the matrix. This behavior dictates the construction of the $m \times n$ matrices with p constant in each column. The `if` statement is required for the single-layer case also because of the behavior of `sum`. If there is only one layer, then the $m \times n$ matrices are all $1 \times n$ row vectors. When given a row vector, `sum` adds its entries to get a single value instead of "summing" the columns of length 1. The `if` statement circumvents this behavior by omitting the `sum` entirely in the single-layer case. The final step, lines 17 through 20, assigns `Inf` to those nonphysical values that were flagged earlier on line 5.

The function *shootray* does not check for consistency of the inputs to ensure that `v` and `z` are column vectors and `p` is a row vector. Also, it lacks flexibility to specify the start and end depths and always shoots the rays from `z(1)` to `z(end)`. *rayfan* traces rays with the same scheme as *shootray* but incorporates error checking and allows the start and end depths to specified. This makes *rayfan* easier to use but slower. *shootray* is intended

Code Snippet 6.11.3 The function *shootray* shoots a fan of rays as defined by the row vector p from z(1) to z(end) through the velocity model defined by the column vectors v and z.

```
function [x,t]=shootray(v,z,p)
... code not displayed ... see shootray.m
iprop=1:length(z)-1;
sn = v(iprop)*p;
[ichk,pchk]=find(sn>1);
cs=sqrt(1-sn.*sn);
vprop=v(iprop)*ones(1,length(p));
thk=abs(diff(z))*ones(1,length(p));
if(size(sn,1)>1)
        x=sum( (thk.*sn)./cs);
        t=sum(thk./(vprop.*cs));
else
        x=(thk.*sn)./cs;
        t=thk./(vprop.*cs);
end
%assign infs
if(~isempty(ichk))
        x(pchk)=inf*ones(size(pchk));
        t(pchk)=inf*ones(size(pchk));
end
```

End Code

velocitycode / shootrayex.m

to be used within a larger ray-tracing scheme, while *rayfan* is more easily used from the command line.

The two-point ray-tracing functions *traceray_pp*, *traceray_ps*, and *traceray* all work similarly with an iterative scheme involving multiple calls to *shootray*. The codes are quite intricate and will not be displayed. *traceray_pp* and *traceray_ps* are constructed to trace P–P and P–S primary reflections and nothing else. *traceray* is more general and can trace an arbitrary ray that has multiple bounces (reflections) and mode conversions. All three codes use similar schemes of building an equivalent layered model that allows the problem to be addressed by shooting rays in one direction only. For example, the P–P reflection problem requires a ray to be traced down through a set of layers to a specified depth and then back up again though nearly the same set of layers. (It will be exactly the same set if the source and receiver depths are the same.) Instead of this two-way problem, a one-way problem is set up by determining the layers for the upgoing leg and placing them beneath the reflector in the order that they are encountered (Figure 6.16). The layers containing the source, receiver, and reflector are adjusted in thickness so that *shootray* can be used to trace rays from z(1) to z(end). A similar equivalent layering can always be constructed for a ray that changes mode (between P and S) and that makes any number of extra bounces. In the former case, the equivalent layering simply must use the correct velocities. In the latter case, the equivalent layering can contain many repeats of a section. This is a technically simple scheme for computing any ray in $v(z)$.

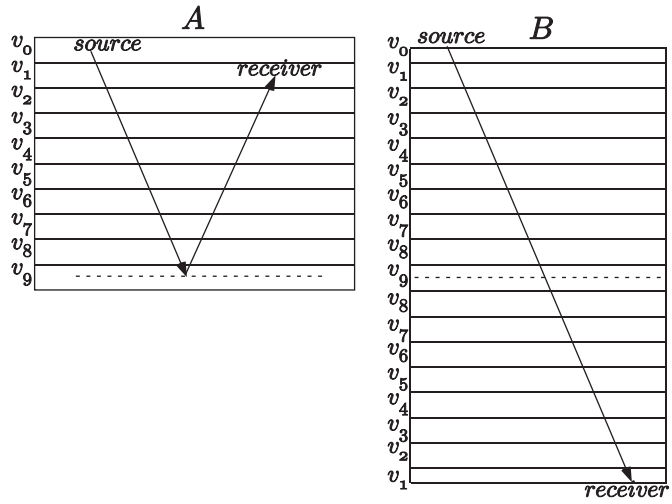


Figure 6.16 (A) A P–P reflection between a source and receiver at different depths. (B) The equivalent one-way raypath to the two-way ray shown in panel A. The dashed line is the reflector in both cases.

The two-point ray-tracing functions can all trace rays from a single starting point to multiple receivers at once. That is, they are *vectorized* to produce simple source gathers. Though the source depth need not equal the receiver depth, if multiple receivers are specified, they must all be at the same depth. This means that geometries such as that of a VSP must be handled by calling the ray tracer separately for each receiver depth. All three programs iterate until all receivers have a ray traced within a *capture radius* of their position or until a maximum number of iterations is reached. The capture radius is the radius of an imaginary circle around each receiver which, if a ray falls within it, is considered "good enough" for that receiver. If the flag `optflag` is set to 1, then the actual traveltime and ray parameter returned are linearly interpolated between the captured ray and the next closest ray. If `optflag` is zero, then the captured ray is used directly.

Unlike many ray tracers, the codes here can reflect, or mode convert, a ray at any depth, not just at a layer boundary. Though this may not be physically correct, it allows the common (and very practical) practice of separating the reflectivity from the propagation model. The idea is that the rays are traced through a simplified *background medium*, which is chosen to give sufficiently accurate traveltimes for the task at hand. Reflections are assumed to come from a level of detail beyond that required for the traveltime calculations. For example, a linear variation of $v(z)$ may be used, with locally determined constants, as a background velocity function. Of course, a user who disagrees with this approach is free to prescribe reflections at actual velocity layer boundaries.

Code Snippet 6.11.4 exhibits the use of *`traceray_pp`* and *`traceray_ps`* to model P–P and P–S reflections with the universal velocity function as the background medium. The results are shown in Figures 6.17a and 6.17b. Line 1 defines the v_P and v_S velocity models with a v_P/v_S of 2. Lines 2 and 3 establish the basic geometry of the source, receivers, and reflector. The capture radius is set to 10% of the receiver spacing and the maximum number

Code Snippet 6.11.4 This example raytraces a P–P and a P–S reflection in a linear-gradient medium. It creates Figures 6.17a and 6.17b.

```
1   zp=0:10:4000;vp=1800+.6*zp;vs=.5*vp;zs=zp;% velocity model
2   zsrc=100;zrec=500;zd=3000;%source receiver and reflector depths
3   xoff=1000:100:3000;caprad=10;itermax=4;%offsets, cap rad, max iter
4   pfan=-1;optflag=1;pflag=1;dflag=2;% default ray fan, and flags
5   % create P-P reflection
6   figure;subplot(2,1,1);flipy;
7   [t,p]=traceray_pp(vp,zp,zsrc,zrec,zd,xoff,caprad,pfan,itermax,...
8      optflag,pflag,dflag);
9   title(['Vertical gradient simulation, P-P mode zsrc=' ...
10     num2str(zsrc) ' zrec=' num2str(zrec)])
11  line(xoff,zrec*ones(size(xoff)),'color','b','linestyle','none',...
12      'marker','v')
13  line(0,zsrc,'color','r','linestyle','none','marker','*')
14  grid;xlabel('meters');ylabel('meters');
15  subplot(2,1,2);plot(xoff,t);grid;
16  xlabel('meters');ylabel('seconds');flipy
17  % P-S reflection
18  figure;subplot(2,1,1);flipy;
19  [t,p]=traceray_ps(vp,zp,vs,zs,zsrc,zrec,zd,xoff,caprad,pfan,...
20     oitermax,ptflag,pflag,dflag);
21  title(['Vertical gradient simulation, P-S mode zsrc=' ...
22     num2str(zsrc) ' zrec=' num2str(zrec)])
23  line(xoff,zrec*ones(size(xoff)),'color','b','linestyle','none',...
24      'marker','v')
25  line(0,zsrc,'color','r','linestyle','none','marker','*')
26  grid;xlabel('meters');ylabel('meters');
27  subplot(2,1,2);plot(xoff,t);grid;
28  xlabel('meters');ylabel('seconds');flipy;
```

End Code

velocitycode /raydemo1.m

of iterations is set to 4, which is adequate for smooth media. The call to *traceray_pp* is on line 7 and the final parameter, `dflag`, instructs the program to draw the rays in the current figure window. Lines 9 and 10 draw symbols indicating the source and receiver, and line 12 plots the traveltime in the bottom half of the figure. Lines 15–22 are very similar except that *traceray_ps* is called to create a P–S reflection. An S–P reflection could be created by reversing the order of the velocity arguments (i.e., `traceray_ps(vs,zs,vp,zp,...)`). Similarly, an S–S reflection can be modeled using *traceray_pp* and providing it with the S-wave velocity functions.

Often, very interesting calculations can be done by calling these functions in a loop. For example, since the receivers are required to be at a constant depth on each call to *traceray_pp*, a VSP cannot be modeled with a single call. However, as Code Snippet 6.11.5 shows, the desired results can be obtained by looping over the receiver depth. Though not as efficient in this mode, the calculation is still simple and accurate. The parameter `pfan` is set to -2 in this case, which causes the ray tracer to start with the final ray

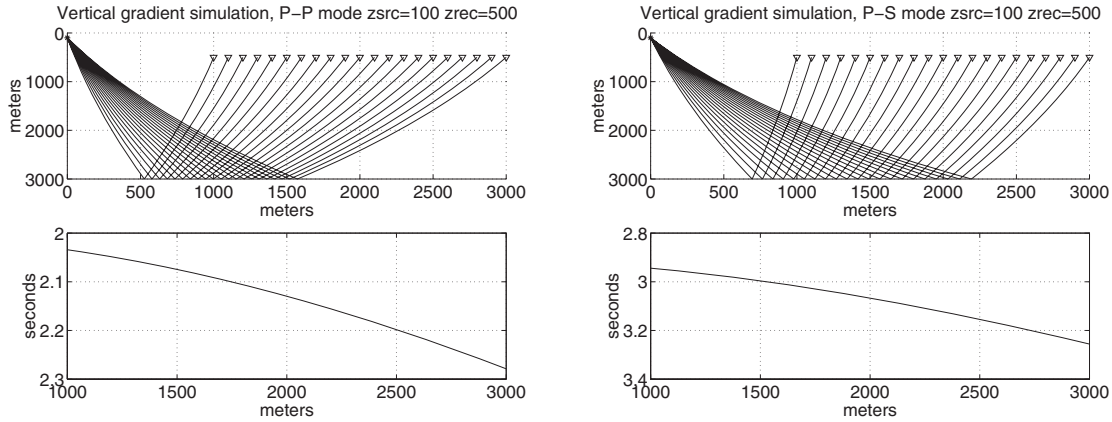


Figure 6.17a (left) A P–P reflection for a background medium of $v_P(z) = 1800 + 0.6z$ m/s. See Code Snippet 6.11.4.
Figure 6.17b (right) A P–S reflection for a background medium of $v_S(z) = 900 + 0.3z$. See Code Snippet 6.11.4.

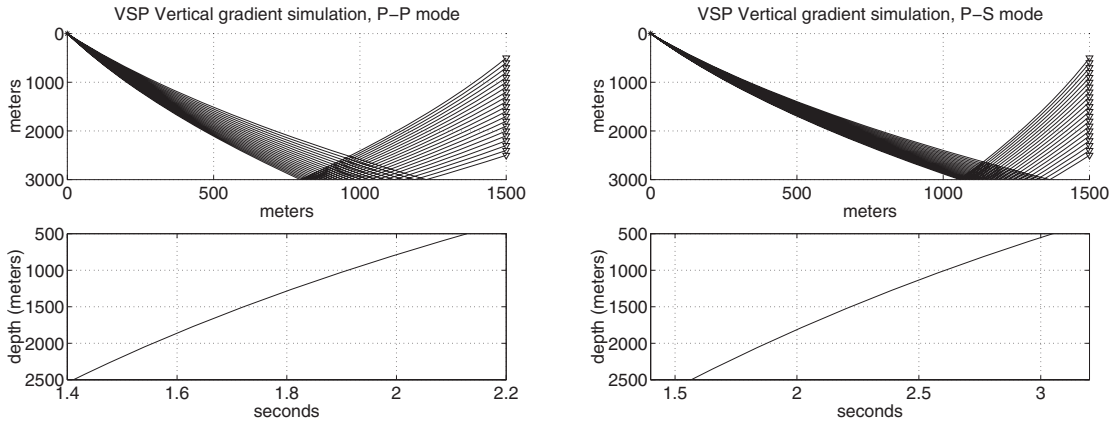


Figure 6.18a (left) A P–P reflection for an offset VSP. See Code Snippet 6.11.5.
Figure 6.18b (right) A P–S reflection for an offset VSP. See Code Snippet 6.11.5.

fan determined the last time it was called. The modeled events are shown in Figures 6.18a and 6.18b.

As a final example of these ray-tracing facilities, consider the calculation of a complicated mode that makes multiple bounces and converts back and forth between P and S modes. This calculation can be done with `traceray` and is shown in Code Snippet 6.11.6. The key idea here is to use a *ray code* that defines the depths and mode types of the different legs of the raypath. A ray code is an $n \times 2$ matrix, with the first column being a list of depths and the second containing either the integer 1 (P-wave) or 2 (S-wave). For example, the ray code `[0 1;1000 2;800 2;1000 1;100 1]` specifies a P-ray from depth 0 to 1000 m. At 1000 m it converts to an S-ray and travels back up to 800 m, and then back down to 1000 m

Code Snippet 6.11.5 This code traces P–P and P–S reflections for an offset VSP by looping over receiver depth. It creates Figures 6.18a and 6.18b.

```
1   zp=0:10:4000;vp=1800+.6*zp;vs=.5*vp;zs=zp;%velocity model
2   zrec=500:100:2500;zsrc=0;zd=3000;xoff=1500;%geometry
3   caprad=10;itermax=4;%cap radius, and max iter
4   pfan=-2;optflag=1;pflag=1;dflag=2;% default ray fan, and flags
5   figure;subplot(2,1,1);flipy
6   t=zeros(size(zrec));
7   for kk=1:length(zrec)
8           if(kk==1)dflag=-gcf;else;dflag=2;end
9           [t(kk),p]=traceray_pp(vp,zp,zsrc,zrec(kk),zd,xoff,...
10        caprad,pfan,itermax,optflag,pflag,dflag);
11  end
12  title([' VSP Vertical gradient simulation, P-P mode '])
13  line(xoff,zrec,'color','b','linestyle','none','marker','v')
14  line(0,zsrc,'color','r','linestyle','none','marker','*')
15  grid;xlabel('meters');ylabel('meters');
16  subplot(2,1,2);plot(t,zrec);
17  xlabel('seconds');ylabel('depth (meters)');grid;flipy;
18
19  figure;subplot(2,1,1);flipy;
20  t=zeros(size(zrec));
21  for kk=1:length(zrec)
22          if(kk==1)dflag=-gcf;else;dflag=2;end
23          [t(kk),p]=traceray_ps(vp,zp,vs,zs,zsrc,zrec(kk),zd,xoff,...
24        caprad,pfan,itermax,optflag,pflag,dflag);
25  end
26  title([' VSP Vertical gradient simulation, P-S mode '])
27  line(xoff,zrec,'color','b','linestyle','none','marker','v')
28  line(0,zsrc,'color','r','linestyle','none','marker','*')
29  grid;xlabel('meters');ylabel('meters');
30  subplot(2,1,2);plot(t,zrec);
31  xlabel('seconds');ylabel('depth (meters)');grid;flipy;
```

End Code

velocitycode /raydemo2.m

while remaining an S-ray. On the second reflection at 1000 m, it converts back to a P-ray and travels up to 100 m, where it ends. The final entry in column 2 is meaningless. Code Snippet 6.11.6 demonstrates this facility by creating a complicated multiple-bounce P-ray (no mode conversions) on line 8 and a multimode ray on line 17. Generally, it might be expected that significant rays like this will be symmetric (i.e., with the same bounce pattern on both the up and the down legs); however, creating an asymmetric ray is perhaps a better demonstration.

There are many more possible ways to use these ray-tracing facilities. For more examples, execute the script *raytrace_demo* and follow the on-screen instructions. The code of this script is similar to the examples presented here and should be comprehensible.

Code Snippet 6.11.6 Here is a demonstration of the use of *traceray* to compute multiple-bounce and multiple-mode raypaths. This code creates Figures 6.19a and 6.19b.

```
zp=0:10:4000;vp=1800+.6*zp;vs=.5*vp;zs=zp;
xoff=1000:100:3000;
caprad=10;itermax=4;%cap radius, and max iter
pfan=-1;optflag=1;pflag=1;dflag=2;% default ray fan, and flags

raycode=[0 1;1500 1;1300 1;2000 1;1800 1;3000 1;2000 1;2300 1;...
    1000 1; 1500 1; 0 1];
figure;subplot(2,1,1);flipy
[t,p]=traceray(vp,zp,vs,zs,raycode,xoff,caprad,pfan,itermax,...
    optflag,pflag,dflag);
title('A P-P-P-P-P-P-P-P-P-P mode in vertical gradient media');
xlabel('meters');ylabel('meters')
line(xoff,zeros(size(xoff)),'color','b','linestyle','none',...
    'marker','v')
line(0,0,'color','r','linestyle','none','marker','*');grid
subplot(2,1,2);plot(xoff,t);
grid;flipy;xlabel('offset');ylabel('time')

raycode=[0 1;1500 2;1300 2;2000 2;1800 2;3000 1;2000 1;2300 1;...
    1000 1; 1500 2; 0 1];
figure;subplot(2,1,1);flipy
[t,p]=traceray(vp,zp,vs,zs,raycode,xoff,caprad,pfan,itermax,...
    optflag,pflag,dflag);
title('A P-S-S-S-S-P-P-P-P-S mode in vertical gradient media');
xlabel('meters');ylabel('meters')
line(xoff,zeros(size(xoff)),'color','b','linestyle','none',...
    'marker','v')
line(0,0,'color','r','linestyle','none','marker','*');grid
subplot(2,1,2);plot(xoff,t);
grid;flipy;xlabel('offset');ylabel('time')
```

End Code

velocitycode /raydemo3.m

Exercises

6.11.4 Consider the velocity model defined by

$$v_P(z) = \begin{cases} 1860 + 0.6z, & 0 \le z < 1000\,\text{m}, \\ 3100 + 0.4(z - 1000), & 1000 \le z < 1550\,\text{m}, \\ 2750 + 0.7(z - 1550), & 1550 \le z, \end{cases}$$

and a v_P/v_S of 2. Create a display showing the P–P and P–S raypaths for source–receiver offsets from 10 to 3010 at 100 m intervals for a reflector at a depth of 2500 m. Let the source and receivers be at $z = 0$. Make another plot showing the

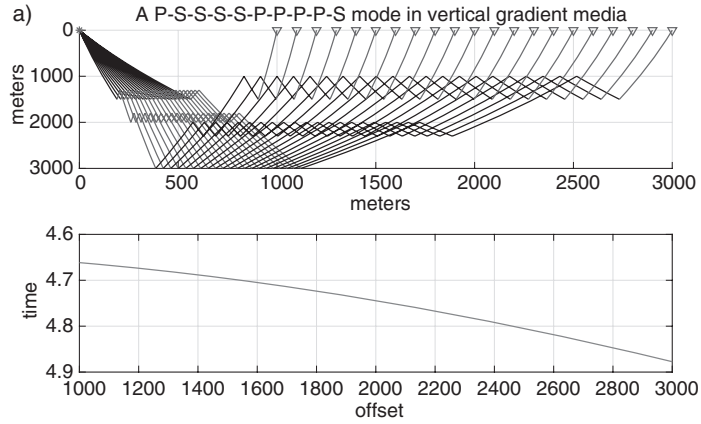


Figure 6.19a A complicated multiple that remains a P-ray throughout. See Code Snippet 6.11.6.

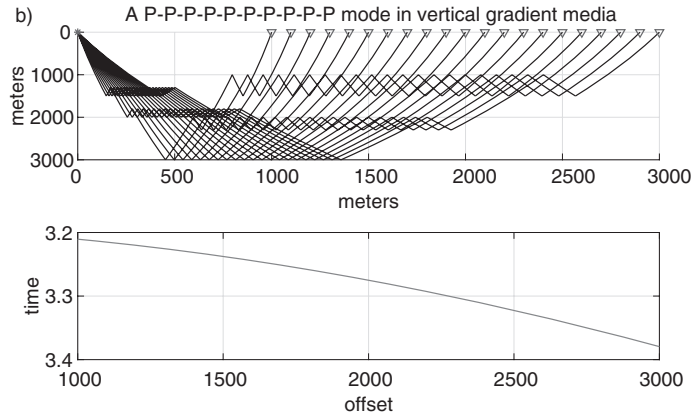


Figure 6.19b A complicated ray that converts back and forth from P to S as it bounces. See Code Snippet 6.11.6.

angle of incidence on the reflector (in degrees) versus offset for both P–P and P–S reflections.

6.11.5 For the velocity model in the previous exercise, what is the largest offset that can be successfully raytraced for the reflector at $z = 2500\,\text{m}$ for both P–P and P–S reflections? Explain why the ray tracing fails. How do these conclusions change if the reflector is at 1500 m?

6.12 Ray Tracing for Inhomogeneous Media

Ray tracing in more general settings than the $v(z)$ method described for the previous setting can be done by detailing the velocity field on a grid in two or three dimensions and then shooting rays through it. The ray trajectories are found by solving a certain differential equation that can be derived from the wave equation. An alternative to the gridded velocity

model is to specify the geometry of geologic interfaces between different velocity units (layers) and then to prescribe the velocity of each unit by a simple analytic function (e.g., constant-velocity layers). Rays are then traced through the model by using analytic results within the layers and explicit Snell's law calculations at the interfaces. The first approach is generally simpler because the ray tracing can be reduced to the solution of an ordinary differential equation that implicitly incorporates Snell's law. In the second approach, the description of the geologic interfaces must be done with great care to ensure that they connect without overlap or voids, and this can require complex geometric calculations. Furthermore, it is generally a complex matter to determine the next interface that a ray will encounter. However, for three-dimensional simulations, the gridded-model approach can rapidly consume a great deal of computer memory and can rapidly become impractical. This is especially true if rays are to be traced through a complex structure that lies far beneath a simple one. In this case, the grid must be made quite small for the complex structure, which results in too much detail for the upper medium. In this book, only the gridded approach in two dimensions will be explored.

6.12.1 The Ray Equation

A reasonable expectation for inhomogeneous media is that individual temporal frequencies can be represented with a mathematical form that is similar to that for a Fourier plane wave. In two or three dimensions, a Fourier plane wave has the form $Ae^{2\pi i(ft\pm\vec{k}\cdot\vec{x})}$, where $\vec{k}$ and $\vec{x}$ are the wavenumber and position vectors. By analogy, for the variable-velocity scalar wave equation

$$\nabla^2\psi - \frac{1}{v^2(\vec{x})}\frac{\partial^2\psi}{\partial t^2} = 0, \tag{6.68}$$

an approximate plane-wave solution will now be assumed in the form

$$\psi(\vec{x},t) = A(\vec{x})e^{2\pi if(t-T(\vec{x}))}, \tag{6.69}$$

where $A(\vec{x})$ and $T(\vec{x})$ are unknown functions describing amplitude and traveltime that are expected to vary with position. In the constant-velocity limit, A becomes constant while T still varies rapidly, which leads to the expectation that variation in A will often be negligible compared with variation in T. Substitution of Eq. (6.69) into Eq. (6.68) will require computation of the Laplacian of the assumed solution. This is done as

$$\nabla^2\psi(\vec{x},t) = \vec{\nabla}\cdot\vec{\nabla}\left[A(\vec{x})e^{2\pi if(t-T(\vec{x}))}\right] = \vec{\nabla}\cdot\left[e^{2\pi if(t-T)}\,\vec{\nabla}A - 2\pi iAe^{2\pi if(t-T)}\,\vec{\nabla}T\right], \tag{6.70}$$

which can be expanded as

$$\nabla^2\psi(\vec{x},t) = \left\{\nabla^2A - 4\pi if\vec{\nabla}A\cdot\vec{\nabla}T - 4\pi^2f^2A\left[\vec{\nabla}T\right]^2 - 2\pi ifA\,\nabla^2T\right\}e^{2\pi if(t-T(\vec{x}))}. \tag{6.71}$$

Then, using this result and $\partial_t^2\psi(\vec{x},t) = -4\pi^2f^2Ae^{2\pi if(t-T(\vec{x}))}$ in Eq. (6.68) and equating real and imaginary parts gives the two equations

$$\left[\vec{\nabla}T\right]^2 - \frac{\nabla^2A}{4\pi^2f^2A} - \frac{1}{v(\vec{x})^2} = 0 \tag{6.72}$$

and

$$\frac{A}{2}\nabla^2 T - \vec{\nabla} A \cdot \vec{\nabla} T = 0. \tag{6.73}$$

So far, these results are exact and no simplification has been achieved. However, the second term in Eq. (6.72) is expected to be negligible when velocity gradients are weak or when frequencies are high, regardless of the velocity gradient. When this term is discarded, the result is the nonlinear partial differential equation called the *eikonal equation*,

$$\left[\vec{\nabla} T\right]^2 - \frac{1}{v(\vec{x})^2} = 0. \tag{6.74}$$

Though not exact, for many realistic situations, a solution to the eikonal equation gives accurate traveltimes through complex media. Equation (6.73) is called the *geometrical spreading equation* because its solution can be shown to describe the flow of energy along a raypath.

The differential equation for raypaths is the vector equation that is implied by the eikonal equation (6.74). For isotropic media, raypaths are normal to the wavefronts, and the latter are described as surfaces where $T(\vec{x}) = \text{constant}$. Therefore, $\vec{\nabla} T$ must be a vector that points in the direction of the raypath, and the eikonal equation shows that $\left|\vec{\nabla} T\right| = v^{-1}$. Consider a wavefront at time $T_1(\vec{x}_1)$ and another at a slightly later time $T_2(\vec{x}_2)$, where $T_2 - T_1$ is very small. Let P_1 be a point on the surface T_1 and P_2 be the corresponding point on T_2 along the normal from P_1 (Figure 6.20). Then $\Delta s = (T_2(P_2) - T_1(P_1))v(P_1)$ is an estimate of the perpendicular distance between these wavefronts, and

$$\vec{\nabla} T(P_1) = \lim_{P_2 \to P_1} \frac{T_2(P_2) - T_1(P_1)}{\Delta s}\hat{s} = \frac{\hat{s}}{v(P_1)}, \tag{6.75}$$

where $\hat{s}$ is a unit vector that is normal to the surface $T_1(P_1)$ or, in other words, $\hat{s}$ points along the raypath. If $d\vec{x}$ is the differential vector pointing from P_1 to P_2, then $\hat{s}$ may be written

$$\hat{s} = \frac{d\vec{x}}{ds}, \tag{6.76}$$

where $ds = |d\vec{x}|$ and s is the arclength along the raypath. Combining Eqs. (6.75) and (6.76) gives the *raypath equation*,

$$\vec{\nabla} T(\vec{x}) = \frac{\hat{s}}{v(\vec{x})} = \frac{1}{v(\vec{x})}\frac{d\vec{x}}{ds}. \tag{6.77}$$

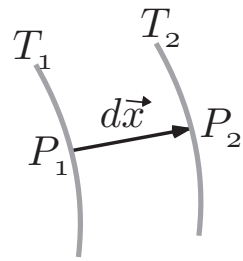


Figure 6.20 A differential raypath element $d\vec{x}$ extends from a point P_1 on a wavefront at time T_1 to a point P_2 on a wavefront at time T_2. The ray segment is perpendicular to both wavefronts and has length $\left|d\vec{x}\right| = \Delta s$.

The traveltime T can be eliminated from Eq. (6.77) by calculating its derivative with respect to arclength, which is

$$\frac{d\,\vec{\nabla}T}{ds} = \frac{d}{ds}\frac{1}{v}\frac{d\vec{x}}{ds}. \tag{6.78}$$

The left-hand side of this equation can be simplified as follows:

$$\frac{d\,\vec{\nabla}T}{ds} = \vec{\nabla}\left[\frac{dT}{ds}\right] = \vec{\nabla}\left[\frac{1}{v(\vec{x})}\right]. \tag{6.79}$$

The last step in this equation is justified using Eq. (6.77) and the identity $d/ds = \hat{s}\cdot\vec{\nabla}$. Intuitively, this step is valid because $\vec{\nabla}T$ points along the raypath and therefore dT/ds, that is, the scalar derivative with respect to arclength along the raypath, gives the full magnitude of $\vec{\nabla}T$. Thus the ray equation is recast as

$$\frac{1}{v^2(\vec{x})}\vec{\nabla}v(\vec{x}) = -\frac{d}{ds}\frac{1}{v(\vec{x})}\frac{d\vec{x}}{ds}, \tag{6.80}$$

which is a second-order ordinary differential equation for the raypath vector $\vec{x}$. This result can be further recast into a system of first-order equations by modifying the right-hand side using

$$\frac{d}{ds} = \frac{dt}{ds}\frac{d}{dt} = \frac{1}{v(\vec{x})}\frac{d}{dt} \tag{6.81}$$

and defining the slowness vector, $\vec{p} = \vec{\nabla}T$, that is (from Eq. (6.77)),

$$\vec{p} = \frac{1}{v(\vec{x})}\frac{d\vec{x}}{ds} = \frac{1}{v^2(\vec{x})}\frac{d\vec{x}}{dt}. \tag{6.82}$$

These last two equations allow Eq. (6.80) to be recast as the first-order system

$$\frac{d\vec{x}}{dt} = v^2(\vec{x})\vec{p} \tag{6.83}$$

and

$$\frac{d\vec{p}}{dt} = -\frac{\vec{\nabla}v(\vec{x})}{v(\vec{x})}. \tag{6.84}$$

Verification that these two equations (6.83) and (6.84) are equivalent to Eq. (6.80) can be done by solving Eq. (6.83) for $\vec{p}$ and substituting it into Eq. (6.84).

Example 6.1 As an illustration of the validity of Eqs. (6.83) and (6.84), it is instructive to show that they reduce to the $v(z)$ case considered previously. If $v(\vec{x}) = v(z)$, then, in two dimensions, let $\vec{x} = [x, z]$ and $\vec{p} = [p_x, p_z]$ so that Eq. (6.84) becomes

$$\frac{dp_x}{dt} = 0 \quad \text{and} \quad \frac{dp_z}{dt} = -\frac{1}{v(z)}\frac{\partial v(z)}{\partial z} \tag{6.85}$$

and Eq. (6.83) is

$$\frac{dx}{dt} = v^2(z)p_x \quad \text{and} \quad \frac{dz}{dt} = v^2(z)p_z. \tag{6.86}$$

The first of the equations (6.85) immediately integrates to $p_x =$ constant. Using this result in the first of the equations (6.86) together with $v^{-2}(z)\, dx/dt = v^{-1}(z)\, dx/\, ds$ results in

$$p_x = \frac{1}{v(z)}\frac{dx}{ds} = \frac{\sin\theta}{v(z)} = \text{constant}, \tag{6.87}$$

where $\sin\theta = dx/ds$ has been used. Of course, this is Snell's law for the $v(z)$ medium. Of the remaining two equations in the system (6.85) and (6.86), the first is redundant because $p_x^2 + p_z^2 = v^{-2}(z)$, and the second gives

$$dt = \frac{dz}{v^2(z)p_z} = \frac{dz}{v(z)\cos\theta} = \frac{dz}{v(z)\sqrt{1 - v^2(z)p_x^2}}. \tag{6.88}$$

When integrated, this will result in Eq. (6.57). In that equation, p is the same as p_x here.

6.12.2 A MATLAB Ray Tracer for $v(x, z)$

The numerical solution of Eqs. (6.83) and (6.84) can be done using well-tested and very general methods for solving first-order ordinary differential equations. An example is the fourth-order Runge–Kutta (RK4) method (Press et al. (1992), Chapter 16), which will be used here. This and other general methods are available in MATLAB; however, it is often useful to implement a specific Runge–Kutta scheme to gain improved flexibility.

To proceed, we define the abstract *ray vector* $\vec{r} = [\vec{x},\ \vec{p}]$. That is, in n dimensions $\vec{r}$ is an $2n$-dimensional vector formed by concatenating the position and slowness vectors. In two dimensions, the ray vector is $\vec{r} = [x\, z\, p_x\, p_z]$ and the time derivative of $\vec{r}$ is defined through Eqs. (6.83) and (6.84) as

$$\frac{dr(1)}{dt} = v^2 r(3), \quad \frac{dr(2)}{dt} = v^2 r(4), \quad \frac{dr(3)}{dt} = -\frac{\partial \ln v}{\partial x}, \quad \frac{dr(4)}{dt} = -\frac{\partial \ln v}{\partial z}. \tag{6.89}$$

Defining the vector $\vec{a} = [v^2\vec{p} - \vec{\nabla}(\ln v)]$, Eq. (6.89) becomes

$$\frac{d\vec{r}}{dt} = \vec{a}. \tag{6.90}$$

Code Snippet 6.12.1 This code builds a velocity matrix representing $v(x,z) = 1800 + 0.6z + 0.4x$ and then uses `ode45` to trace a ray. It creates Figure 6.21.

```
1   dg=10; %grid spacing
2   tstep=0:.004:3; %time step vector
3   x=0:dg:10000;z=x'; %x and z coordinates
4   v=1800+.6*(z*ones(1,length(x)))+.4*(ones(length(z),1)*x);%velocity
5   rayvelmod(v,dg);clear v;%initialize the velocity model
6   theta=pi*45/180;%takeoff angle
7   r0=[0,0,sin(theta)/1800,cos(theta)/1800]';%initial value of r
8   [t,r]=ode45('drayvec',tstep,r0);%solve for raypath
9   plot(r(:,1),r(:,2));flipy%plot
```

End Code

velocitycode /rayvxzdemo1.m

Then, given this prescription for $d\vec{r}/dt$ and initial values for $\vec{r}$, the fourth-order Runge–Kutta method is invoked to integrate Eq. (6.90) for $\vec{r}(t)$.

The MATLAB implementation requires a velocity matrix giving $v(x,z)$ on a grid with $\Delta x = \Delta z \equiv \Delta g$ and uses the convention that $(x = 0, z = 0)$ is in the upper left corner. Prior to ray tracing, the function *rayvelmod* is invoked, with its arguments being the velocity matrix and the grid spacing Δg. This function creates a number of global variables that will be used repeatedly in the ray tracing. These include matrices of v^2, $\partial_x \ln v$, and $\partial_z \ln v$ that are needed in Eq. (6.89). This precomputation speeds the ray tracing and is especially beneficial if a great many rays are to be traced; however, the cost is that the memory overhead is three times that of the simple velocity matrix. *rayvelmod* need only be called once at the beginning of the ray tracing unless the velocity model is changed.

The function *drayvec* implements the computation of $d\vec{r}/dt$ according to Eq. (6.89). This function is designed with the interface required by MATLAB's built-in ordinary-differential-equation solver `ode45`. This latter function implements an RK4 scheme by calling a user-designed function such as *drayvec* that computes the time derivative of the solution vector. The general interface required by `ode45` for such a function is that it must have two input arguments that are the current time and the current value of the solution vector $\vec{r}$. Thus, even though Eq. (6.90) does not require the value of t to compute $d\vec{r}/dt$, *drayvec* requires t as input but does not use it.

Code Snippet 6.12.1 illustrates the tracing of a single ray in a medium with both vertical and horizontal velocity gradients. The time step vector is established on line 2 and a 0.004 s time step is used. Smaller time steps will improve accuracy but will also lengthen the computation. For a particular velocity model, some testing may be required to determine the optimal time step for the problem at hand. The velocity matrix is built on line 4 and then passed to *rayvelmod* on line 5. It is then cleared to save space because *rayvelmod* has established the required velocity information in global matrices. The takeoff angle and initial values for $\vec{r}$ are calculated in lines 7 and 8. As mentioned in Exercise 6.12.1, the components of $\vec{p}$ are not independent of one another, since $\vec{p} \cdot \vec{p} = v^{-2}$, and this is illustrated in the calculation of the initial values. Finally, `ode45` is invoked to integrate

Eq. (6.90) and thus compute $\vec{r}$ for the vector of times established on line 2. The calculated ray is shown in Figure 6.21.

In the computation of $d\vec{r}/dt$, it is generally true that the ray coordinates at a particular time will not coincide with grid points in the model. Thus the estimation of the velocity terms in Eqs. (6.89) requires some form of interpolation. By default, `drayvec` uses nearest-neighbor interpolation because this is the fastest method. For more accuracy, bilinear interpolation is also available and its use is controlled through a global variable that is explained in the `drayvec` help file.

A problem with the use of `ode45` that is apparent upon close inspection of Figure 6.21 is that the ray has been traced beyond the bounds of the velocity model. To avoid indexing into the velocity array beyond its bounds, `drayvec` has been constructed to use the first (or last) column to represent a simple $v(z)$ medium for rays that go beyond the beginning (or end) of the model. A similar strategy is used to cope with rays that go beyond the minimum or maximum depth. Though this allows the solution to proceed, a better approach would be to detect when a ray has reached the boundary of the model and stop the ray tracing. For this purpose, an RK4 solver has been built as described in Press et al. (1992) and is incorporated in the functions `shootrayvxz` and `shootrayvxz_g`. In these functions, the ray is tested at each time step to determine if it is within the model bounds, and ray tracing is halted before the maximum time if the ray leaves the model. The syntax to trace a ray with either program is essentially the same as with Code Snippet 6.12.1 except that the command `[t,r]=shootrayvxz(tstep,r0)` replaces `[t,r]=ode45('drayvec',tstep,r0)` on line 8. The functions `shootrayvxz` and `shootrayvxz_g` differ in that the latter calls `drayvec` to compute $d\vec{r}/dt$, while the former does this computation directly without the function call. The result is that `shootrayvxz` is much more efficient but does not offer bilinear interpolation as does `shootrayvxz_g`. If nearest-neighbor interpolation is satisfactory, then `shootrayvxz` should be preferred because it is much faster.

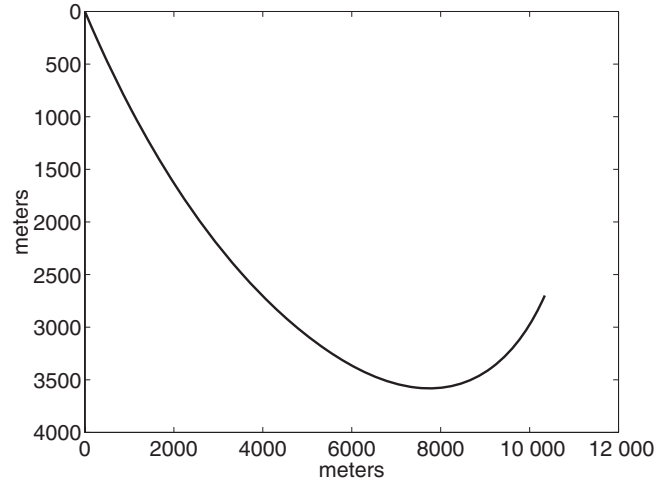


Figure 6.21 A ray traced through $v(x, z) = 1800 + 0.6z + 0.4x$ using `ode45`, as shown in Code Snippet 6.12.1.

Another useful ray tracer is *shootraytosurf*. This function works similarly to *shootrayvxz* but the ray is traced until it reaches $z = 0$ or a maximum time limit is exceeded. This is useful in normal-incidence raytrace modeling, which will be discussed in Chapter 7 in connection with normal-incidence raytrace migration.

6.13 Chapter Summary

This chapter has provided a summary of the different kinds of velocity measures used in exploration seismology and defined their interrelationships. These many different measures were all defined by their link to the speed of wave propagation, the instantaneous velocity, and conversions between them were established. In the closing sections, the concept of ray tracing was explored.

7 Elementary Migration Methods

In exploration seismology, *migration* refers to a multichannel processing step that attempts to spatially reposition events and improve focusing. The term is essentially synonymous with *imaging* (though the latter term has a specific secondary meaning as a substep in a migration process). Before migration, seismic data is usually displayed with traces plotted at the surface locations of the receivers and with a vertical time axis. This means that dipping reflections are systematically mispositioned in the lateral coordinate and the vertical time axis needs a transformation to depth. Also problematic is the unfocused nature of seismic data before migration. It is very much like looking through an unfocused camera lens. Just as the construction of a camera lens requires knowledge of the physics of light propagation, migration algorithms incorporate the physics of acoustic and elastic wave propagation. In addition to spatial positioning and focusing, migration algorithms perform amplitude and phase adjustments that are intended to correct for the effects of the spreading (or convergence) of raypaths as waves propagate.

Migration can be viewed as an approximate solution to the general elastic wavefield inversion problem (Gray, 1997). Full elastic inversion uses the entire seismic wavefield as input into a mathematical process that seeks to estimate the elastic parameters of the Earth. This is called an inverse problem because it is opposite to the classical *forward modeling* problem of predicting the seismic wavefield response of a known elastic Earth model. Generally, a forward problem can be reduced to finding the solution to a partial differential equation, in this case the elastic wave equation, given specifications of the coefficients of the equation as functions of position and given appropriate boundary conditions. Though this is often a very difficult process, it is usually more easily accomplished than the corresponding inverse problem. The inverse problem usually amounts to estimating the coefficients of a partial differential equation given its response to a known source. Often, these inverse problems are *ill-posed*, which is a mathematical term for a problem whose inputs are insufficient to determine all of its expected outputs. For example, the solution to the constant-velocity migration problem (Section 7.4.1) requires two surface measurements, the wavefield (ψ) and its vertical derivative ($\partial_z \psi$), all along the surface. Despite this mathematical requirement, there is no feasible technology for measuring $\partial_z \psi$, so methods must be found to deal with only ψ. In addition to being ill-posed, inversion problems are generally nonlinear, and practical schemes are typically linearized approximations. This means that they are very sensitive to an assumed initial model. This is unfortunate because knowledge of the subsurface is very limited and the construction of initial models is extremely difficult.

Thus, for many reasons, migration amounts to a very approximate solution to a general, nonlinear inverse problem. The need to find approximate solutions has led to a large number of different migration algorithms that are generally distinguished by the kind of

approximations made. As a result there are many different, overlapping ways to categorize migration algorithms. For example, it can be shown that an assumption that waves are traveling only upward at the Earth's surface, though physically incorrect, allows the solution to proceed without knowledge of $\partial_z \psi$. (Or, equivalently, this assumption allows $\partial_z \psi$ to be calculated from ψ.) This approach is called the *one-way wave* assumption and is very common. Given that one-way waves will be considered, there are finite-difference algorithms that offer great flexibility with spatial variations of velocity but suffer from *grid dispersion* and *angle limitations*, Kirchhoff methods that can easily accommodate arbitrary variations in seismic recording geometry but require ray tracing to guide them, and Fourier (or spectral) techniques that offer high fidelity but have difficulty coping with rapid variations in either velocity or recording geometry. All of these techniques attempt, in some manner, to *downward continue* measurements made at $z = 0$ into the subsurface. There are also useful distinctions about whether the estimation of spatial reflectivity $r(x, y, z)$ is made directly from $z = 0$, by *direct* methods, or whether $r(x, y, z)$ is estimated from $z - \Delta z$, by *recursive* methods. Finally, perhaps the most common categorization comes under the confusing labels of *time migration* and *depth migration*. The former is an earlier technology that remains viable because it is very insensitive to velocity errors, even though it is known to be accurate only if $\partial_x v \sim 0$. On the other hand, depth migration is the only currently available imaging technology that is accurate when $\partial_x v$ varies strongly; however, it requires a very accurate specification of the velocity model.

These last remarks hint at the general chicken-and-egg nature of modern migration methods. A true inversion technique would derive the velocity model from the data as part of the inversion. Migration requires the velocity model as input and merely attempts to reposition, focus, and adjust amplitudes. These really all turn out to be the same thing. In order to be a useful process, it should be true that migration requires only an approximate, or background, velocity model and that more detailed velocity information can be extracted from a successful migration than was input into it. This is generally the case, though it can be very difficult to achieve in structurally complex areas. Especially for depth migration, the construction of the velocity model is the central difficulty in achieving a successful result.

This chapter will discuss "elementary" migration methods. This refers to techniques designed to migrate stacked seismic data (poststack migration) in the case of simple velocity variations.

7.1 Stacked Data

7.1.1 Band-Limited Reflectivity

The ultimate goal of a migration is to transform the seismic data into *band-limited reflectivity*. In the simplest context, reflectivity means the *normal-incidence reflection coefficient* of P-waves. Potentially, every point in the subsurface has a reflectivity value associated with it. In a one-dimensional layered medium, the P-wave reflectivity is given by $r_k = 2(I_k - I_{k-1})/(I_k + I_{k-1})$, where $I_k = \rho_k v_k$ is the P-wave impedance of the kth

layer. In the continuous case, this can be written as $r_{\delta z}(z) = 0.5\,\partial_z \ln(I(z))\,\delta z$, where δz is a small increment of depth. In a 3D context, the reflectivity of a small Earth element, δ vol, is conveniently described by

$$r(x,y,z) = 0.5\left|\vec{\nabla}(\log(I(x,y,z)))\right|\ \delta \text{ vol}. \tag{7.1}$$

At best, Eq. (7.1) can only be an expression for the normal-incidence reflectivity. More generally, the reflectivity must be acknowledged to be a function of the angle of incidence as well as of spatial position. For the case of plane elastic waves in layered media, the Zoeppritz equations (Aki and Richards, 1980) are an exact prescription of this variation. The Zoeppritz equations are highly complex and nonlinear, and the estimation of their approximate parameters from seismic data is called the study of *amplitude variation with offset*, or AVO. (A nearly synonymous term is *amplitude variation with angle*, or AVA.) Such complex reflectivity estimates are key to true lithology estimation and are a subject of much current research. Ideally, AVO analysis should be conducted simultaneously with migration or after migration. For the current discussion, it is assumed that only the normal-incidence reflectivity, $r(x,y,z)$, is of interest and that a stacked seismic section provides a band-limited estimate of $r(x,y,z)$.

The estimate of reflectivity must always be band limited to some *signal band* of temporal frequencies. This is the frequency range over which signal dominates over noise. Unless some sort of nonlinear or model-based inversion is done, this signal band is determined by the power spectrum of the source, the data fold,[1] the types of coherent and random noise, and the degree of anelastic attenuation (Q loss). The optimal stacked section will have a zero-phase, white embedded wavelet. This means that it should obey the simple convolutional model

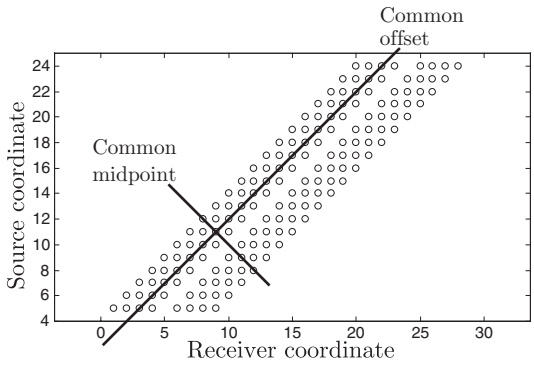
$$s(t) = r(t) \bullet w(t), \tag{7.2}$$

where $w(t)$ is a zero-phase wavelet whose amplitude spectrum is white over some limited passband. If $w(t)$ has residual phase or spectral color, then these will adversely affect the resolution of the final migrated image. They should be dealt with before migration.

7.1.2 The Zero-Offset Section

A discussion of poststack migration greatly benefits from a firm theoretical model of the CMP stack. A simple model of stacked seismic data is the *zero-offset section*, or "ZOS," model. This model asserts that the prestack processing and CMP stack estimate a signal-enhanced version of what we would have recorded had there been a single, coincident source/receiver pair at each CMP. Each stacked trace represents a separate physical experiment that can be approximately described by the scalar wave equation (assuming acoustic energy only). Taken together, the ensemble of stacked traces has a more complicated basis in physical theory.

[1] The *fold* of seismic data applies to stacked data and refers to the number of elementary traces that are summed together (stacked) to create one stacked trace. For example, if the fold is 100, then each stacked trace is the sum of 100 traces from the unstacked dataset.



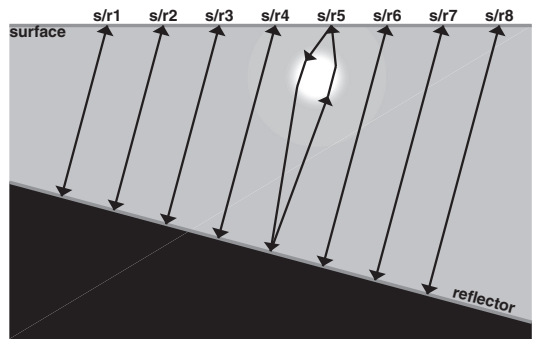


Figure 7.1a (left) A 2D seismic line is depicted in the (s, g) plane. Each shot is represented as having eight receivers with a central gap. Common-midpoint and common-offset coordinates x and h are depicted.

Figure 7.1b (right) Most, but not all, zero-offset raypaths are also normal-incidence raypaths. At positions 1–4 and 6–7, the only possible zero-offset reflection raypaths are also normal incidence. At position 5, the velocity lens makes possible a nonnormal-incidence, but still zero-offset raypath. A normal-incidence path is still possible owing to the symmetry of the lens but is not shown. Shading indicates velocity, with darker being faster. The symbol "s/r" indicates a coincident source–receiver pair.

The major steps in the estimation of the signal-enhanced ZOS are:

- *Spherical spreading correction*: a time-dependent amplitude scaling that approximately corrects for spherical divergence.
- *Deconvolution:* the source signature and the average Q effect are estimated and removed.
- *Sort to* (x, h) *coordinates:* the data is considered to be gathered in (s, g) (source, geophone) coordinates and then sorted according to midpoint, x, and half-offset, h (Figure 7.1a).
- *Statics correction:* near-surface static time delays are estimated and removed.
- *NMO removal:* the data is taken through stacking velocity analysis and the resulting velocities are used to remove normal-moveout (NMO) time delays.
- *Residual statics correction:* residual time delays (small) are sought and removed. Usually this is a surface-consistent step.
- *Trace balancing:* this might be considered optional, but some step is needed to correct for source strength and geophone coupling variations.
- *CMP stacking:* all traces with a common midpoint coordinate are summed. Events which have been flattened on CMP gathers are enhanced. Other events and random noise are reduced.

The ZOS is said to be signal enhanced because the stacking process has been designed to select against multiples. The simplest (and dominant) raypath which returns energy to its source is called the *normal-incidence* raypath. Quite obviously, the normal-incidence reflection will send energy back up along the path that it traveled down on. However, it is possible to have zero-offset paths that are not normal-incidence paths (Figure 7.1b).

7.1.3 The Spectral Content of the Stack

The sort from (s, g) to (x, h) marks the transition from single-channel to surface-consistent processing. Where it actually takes place is somewhat arbitrary because a seismic processing system can always re-sort the data whenever required. However, it is a logical transition that must occur somewhere before stack. (x, h) coordinates are defined by

$$x = \frac{s+g}{2} \quad \text{and} \quad h = \frac{g-s}{2}, \tag{7.3}$$

for which the inverse transformation is

$$s = x - h \quad \text{and} \quad g = x + h. \tag{7.4}$$

The relationship between (s, g) and (x, h) is depicted graphically in Figure 7.1a.

Mathematically, a 2D prestack seismic dataset is a 3D function, $\psi(s, g, t)$. This dataset can be written as an inverse Fourier transform of its spectrum through

$$\psi(s, g, t) = \int_{V_\infty} \phi(k_s, k_g, f) e^{2\pi i(k_s s + k_g g - ft)}\, dk_s\, dk_g\, df, \tag{7.5}$$

where the notation V_∞ indicates that the integration covers the entire relevant, possibly infinite, portion of (k_s, k_g, f) space. Imposing the coordinate transform of Eq. (7.4) gives

$$\psi(x, h, t) = \int_{V_\infty} \phi(k_s, k_g, f) e^{2\pi i(k_s(x-h) + k_g(x+h) - ft)}\, dk_s\, dk_g\, df, \tag{7.6}$$

or

$$\psi(x, h, t) = \int_{V_\infty} \phi(k_s, k_g, f) e^{2\pi i((k_s+k_g)x + (k_s-k_g)h - ft)}\, dk_s\, dk_g\, df. \tag{7.7}$$

This motivates the definitions

$$k_x = k_s + k_g \quad \text{and} \quad k_h = k_s - k_g, \tag{7.8}$$

or the inverse relationship

$$k_s = \frac{k_x + k_h}{2} \quad \text{and} \quad k_g = \frac{k_x - k_h}{2}. \tag{7.9}$$

This allows Eq. (7.7) to be written

$$\psi(x, h, t) = \frac{1}{2} \int_{V_\infty} \phi(k_x, k_h, f) e^{2\pi i(k_x x + k_h h - ft)}\, dk_x\, dk_h\, df, \tag{7.10}$$

where the factor of $\frac{1}{2}$ comes from the *Jacobian* of the coordinate transformation given in Eq. (7.9). Both Eqs. (7.5) and (7.10) are proper inverse Fourier transforms, which

shows that the wavenumber relationships given in Eqs. (7.8) and (7.9) are the correct ones corresponding to the (s, g)-to-(x, h) coordinate transformation.

Quite a bit can be learned from Eq. (7.9) about how the spectral content of the prestack data gets mapped into the poststack data. The maximum midpoint wavenumber, $k_{x\,\max}$, comes from the sum of $k_{s\,\max}$ and $k_{g\,\max}$. As will be shown formally later, the maximum wavenumber is directly proportional to the horizontal resolution (i.e., the greater the wavenumber, the better the resolution). The meaning of the maximum wavenumber is that it is the largest wavenumber that contains signal. It is typically less than the Nyquist wavenumber but cannot be greater. Thus $k_{s\,\max} \leq k_{s\,\mathrm{Nyq}} = 1/(2\,\Delta s)$ and $k_{g\,\max} \leq k_{g\,\mathrm{Nyq}} = 1/(2\,\Delta g)$. It is customary to sample the x coordinate at $\Delta x = 0.5\,\Delta g$ so that $k_{x\,\mathrm{Nyq}} = 1/(2\,\Delta x) = 2k_{g\,\mathrm{Nyq}}$, which means that the stacking process will generate k_x wavenumbers up to this value from combinations of k_s and k_g according to Eq. (7.8). However, because spatial antialias filters (source and receiver arrays) are not very effective, it must be expected that wavenumbers higher than Nyquist will be present in both the k_s and the k_g spectra. The only way to generate unaliased wavenumbers up to $k_{x\,\mathrm{Nyq}}$ in the stacking process is if $\Delta s = \Delta g$ so that $k_{x\,\mathrm{Nyq}} = k_{s\,\mathrm{Nyq}} + k_{g\,\mathrm{Nyq}}$. This means that there must be a shotpoint for every receiver station, which is a very expensive acquisition. Normal 2D land shooting puts a shotpoint for every n receiver stations, where $n \geq 3$. This means that k_x wavenumbers greater than a certain limit will be formed from completely aliased k_s and k_g wavenumbers. This limiting unaliased wavenumber is

$$k_{x\,\mathrm{lim}} = \frac{1}{2\,\Delta g} + \frac{1}{2\,\Delta s} = \frac{1}{2\,\Delta g} + \frac{1}{2n\,\Delta g} = \frac{n+1}{2n\,\Delta g}. \tag{7.11}$$

For $n = 3$, $k_{x\,\mathrm{lim}} = \frac{2}{3}k_{x\,\mathrm{Nyq}}$, so that shooting every third group will result in a conventional stack with the upper third of the k_x spectrum being completely useless.

Even if $n = 1$, there will still be aliased contributions to k_x if the k_s and k_g spectra are aliased, as they normally are. For example, $k_{x\,\mathrm{Nyq}}$ can be formed with unaliased data from $k_{s\,\mathrm{Nyq}} + k_{g\,\mathrm{Nyq}}$ but it can also be formed from the sum of $k_s = 0$ and $k_g = 2k_{g\,\mathrm{Nyq}}$, and there are many more such aliased modes. Similarly, wavenumbers less than $k_{x\,\mathrm{lim}}$ can be formed by a wide variety of aliased combinations. Further analysis is helped by having a more physical interpretation for k_s and k_g so that their potential spectral bandwidth can be estimated.

As explained in Section 6.9, the apparent velocity of a wave as it moves across a receiver array is given by the ratio of frequency to wavenumber. For a source record (*common source gather*), this has the easy interpretation that

$$\frac{f}{k_g} = \frac{v_0}{\sin\theta_0}, \tag{7.12}$$

which means that

$$k_g = \frac{f \sin\theta_0}{v_0}. \tag{7.13}$$

So, if $f_{\max}$ is the maximum temporal frequency and $(\sin\theta_0)_{\max} = 1$, then

$$k_{g\,\max} = \frac{f_{\max}}{v_{\mathrm{o}\,\min}}, \tag{7.14}$$

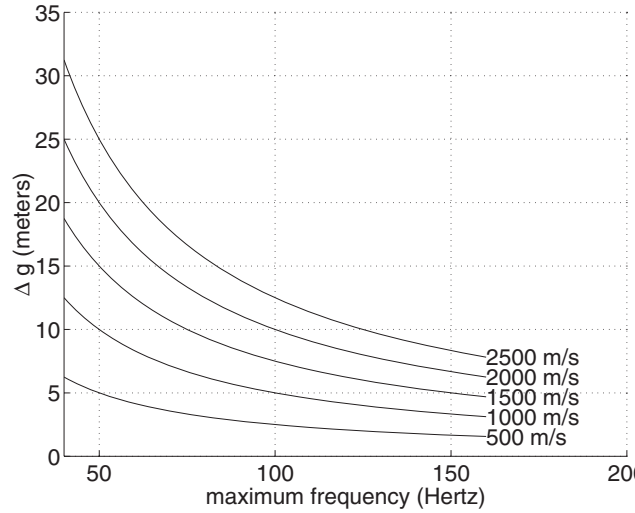


Figure 7.2 These curves show the maximum spatial sample rate required to avoid aliasing of k_g on shot records (assuming point receivers) as a function of the maximum signal frequency. A curve is drawn for each of five near-surface velocities. This is based on Eq. (7.15).

where $v_{\text{o}\,\min}$ is the slowest near-surface velocity found along the receiver spread. So, to avoid any aliasing of k_g, assuming point receivers, the receiver sampling must be such that $k_{g\,\text{Nyq}} \geq k_{g\,\max}$, which leads to the antialiasing condition

$$\Delta g \leq \frac{v_{\text{o}\,\min}}{2f_{\max}}, \tag{7.15}$$

assuming the equality in Eq. (7.15) allows the maximal sampling curves in Figure 7.2 to be calculated. Alternatively, if coarser sampling than suggested by these curves is planned, then a portion of the k_g spectrum will be aliased. An array can be designed to suppress the aliased wavenumbers, though this is not as effective as temporal antialias filtering.

The wavenumbers k_s are those which would be measured by an f–k analysis on a common receiver gather. Reciprocity suggests that a common receiver gather is similar to a common source gather with the source and receiver positions interchanged (Figure 7.3). It cannot be expected that the amplitudes will be completely reciprocal on such gathers, but the traveltimes should be. Since the relationship between apparent velocity and frequency–wavenumber ratios is based on traveltimes, not amplitudes, the arguments just made for the interpretation of k_g also apply to k_s. The interpretation is that f/k_s gives the apparent velocity of a monochromatic wave as it leaves a source such that it will arrive at the common receiver. Thus

$$\frac{f}{k_s} = \frac{v_0}{\sin\alpha_0}, \tag{7.16}$$

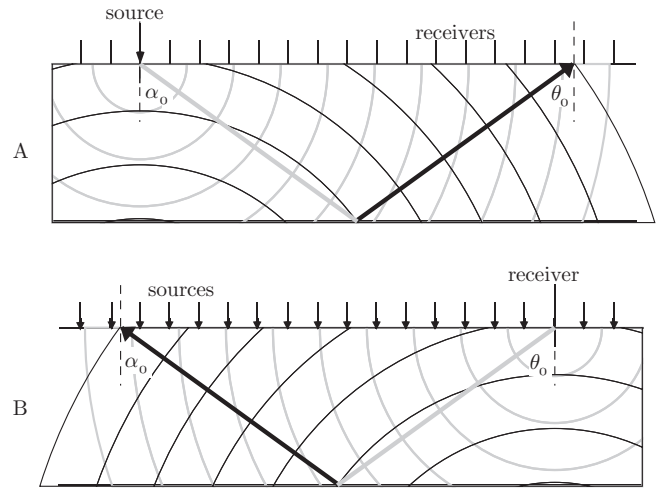


Figure 7.3 (A) The wavenumber k_g is measured on a common source gather and estimates the emergence angle, θ_0, of a monochromatic wave. (B) The wavenumber k_s is measured on a common receiver gather and estimates the takeoff angle α_0 of the ray that arrives at the common receiver.

where α_0 is the *takeoff angle* required to reach the common receiver and v_0 is as before. Comparing Eqs. (7.12) and (7.16) leads to the conclusion that the potential bandwidth of k_s is just as broad as that of k_g.

For the wavenumber k_h, the stacking process rejects all but $k_h = 0$. Stacking can be modeled by an integral over h as

$$\psi_0(x,t) = \int_{\text{all } h} \psi(x,h,t)\, dh. \tag{7.17}$$

If Eq. (7.10) is substituted into Eq. (7.17) and the order of integration reversed, it results that

$$\psi_0(x,t) = \frac{1}{2}\int_{V_\infty} \left[\int_{\text{all } h} e^{2\pi i k_h h}\, dh \right] \phi(k_x, k_h, f) e^{2\pi i(k_x x - ft)}\, dk_x\, dk_h\, df. \tag{7.18}$$

The integral in square brackets is $\delta(k_h)$, which, in turn, collapses the k_h integral by evaluating it at $k_h = 0$ to give

$$\psi_0(x,t) = \frac{1}{2}\int_{V_\infty} \phi(k_x, 0, f) e^{2\pi i(k_x x - ft)}\, dk_x\, df. \tag{7.19}$$

So, the CMP stacking process passes only the $k_h = 0$ wavenumber, which is a very severe f–k filter applied to a CMP gather. It is for this reason that f–k filtering applied to CMP gathers does not typically improve the stack. However, f–k filtering of source and receiver gathers can have a dramatic effect. The $k_h = 0$ component is the average value over the

CMP gather and corresponds to horizontal events. Of course, this is done after normal-moveout removal, which is designed to flatten those events deemed to be primaries.

So far, the discussion has been about wavenumber spectra, with nothing being said about the effect of stacking on temporal frequencies. Bearing in mind the goal of estimating band-limited reflectivity, the ideal stacked section should have all spectral color removed except that attributable to the reflectivity. Most prestack processing flows rely on one or more applications of statistical deconvolution to achieve this. Typically, these deconvolution techniques are not capable of distinguishing signal from noise, and so whiten (and phase rotate) both. If the prestack data input into the stacking process has been spectrally whitened, it will always suffer some attenuation of high frequencies owing to the stacking out of noise. This is because the stacking process improves the signal-to-noise ratio by $\sqrt{\text{fold}}$. Moreover, this effect will be time-variant in a complicated way because of two competing effects. First, the fold in any stacked section is actually time-variant because of the front-end mute. Until the time at which nominal fold is reached, the stacking effect is therefore time-variant. Second, the signal-to-noise ratio in any single prestack trace must decrease with both increasing time and increasing frequency owing to attenuation (i.e., Q effects). As a result, it is virtually guaranteed that the poststack data will need further spectral whitening. It is also highly likely that some sort of wavelet processing will be required to remove residual phase rotations.

7.1.4 The Fresnel Zone

The zero-offset rays shown in Figure 7.1b are only a high-frequency approximation to what really happens. The zero-offset recording at a point on the Earth's surface actually contains scattered waves from a very broad zone on each subsurface reflector. This contrasts with the high-frequency ray-tracing concepts that suggest the reflection occurs at a point. Figure 7.4 illustrates the concept of the zero-offset Fresnel zone. The width, w, of the Fresnel zone is defined as the diameter of a disk whose center is marked by the ray P

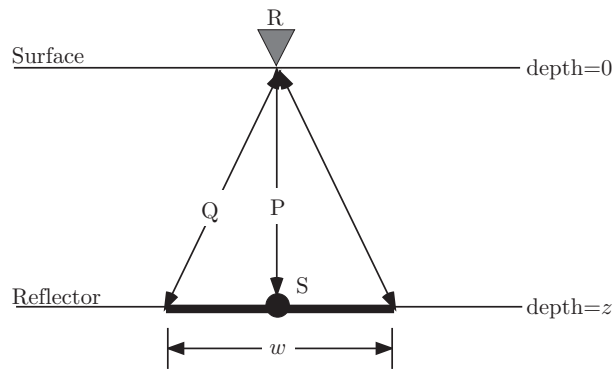
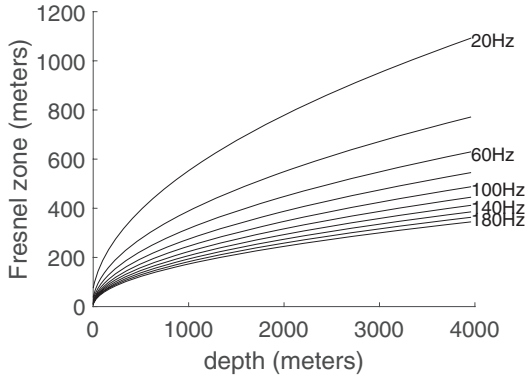


Figure 7.4 The zero-offset *Fresnel zone* is defined as the width of a disk on a subsurface reflector from which scattered energy arrives at R with no more than a $\lambda/2$ phase difference.



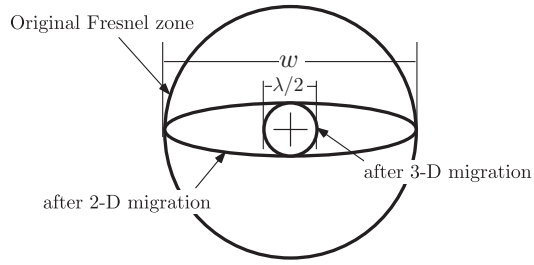


Figure 7.5a (left) The width of the Fresnel zone (based on Eq. (7.21)) versus depth for a variety of frequencies and an assumed velocity of 3000 m/s.

Figure 7.5b (right) A 3D migration collapses the Fresnel zone to a disk of diameter $\lambda/2$, while a 2D migration only collapses the Fresnel zone in the inline direction.

and whose edge is marked by the ray Q. Ray P is a simple zero-offset ray, while ray Q is defined to be $\lambda/4$ longer than ray P. Assuming constant velocity, this means that scattered energy that travels down and back along path Q will be $\lambda/2$ out of phase at the receiver, R, compared with that which travels along path P. All paths intermediate between P and Q will show a lesser phase difference. Therefore, it is expected that path Q will interfere destructively with P, and it is taken to define the diameter of the central disk from which scattered energy shows constructive interference at R. The equation defining the Fresnel zone diameter is therefore

$$\sqrt{z^2 + \left(\frac{w}{2}\right)^2} - z = \frac{\lambda}{4}, \tag{7.20}$$

which is easily solved for w to give

$$w = 2\sqrt{\left(\frac{\lambda}{4} + z\right)^2 - z^2} = \sqrt{2z\lambda}\sqrt{1 + \frac{\lambda}{8z}} = \sqrt{\frac{2zv}{f}}\sqrt{1 + \frac{v}{8fz}}, \tag{7.21}$$

where, in the third expression, $\lambda f = v$ has been used. If $z \gg \lambda$, then the approximate form $w \sim \sqrt{2z\lambda}$ is often used. The size of a Fresnel zone can be very significant in exploration, as it is often larger than the target (Figure 7.5a).

Migration can be conceptualized as lowering the source and receiver to the reflector and so shrinks the Fresnel zone to its theoretical minimum of $\lambda/2$. That the Fresnel zone does not shrink to zero is another way of understanding why reflectivity estimates are always band limited. However, seismic data is a bit more complicated than this in that it possesses signal over a bandwidth, while the Fresnel zone concept is monochromatic. Roughly speaking, the effective Fresnel zone for broadband data can be taken as the average of the individual Fresnel zones for each frequency.

The 3D Fresnel disk collapses, under 3D migration, from a diameter of w to a diameter of $\lambda/2$. However, much seismic data is collected along the lines, so that only 2D migration is possible. In this case, the Fresnel zone only collapses in the inline direction and remains

at its unmigrated size in the crossline direction (Figure 7.5b). Thus the reflectivity shown on a 2D migrated seismic line must be viewed as a spatial average in the crossline direction over the width of the Fresnel zone. This is one of the most compelling justifications for 3D imaging techniques, even in the case of sedimentary basins with flat-lying structures.

7.2 Fundamental Migration Concepts

7.2.1 One-Dimensional Time–Depth Conversions

It is quite common to convert a single trace from time to depth or the reverse, by a simple one-dimensional conversion that is called a *stretch*. This is accomplished as a simple mapping of samples from the time trace to the depth trace. The mapping from time to depth is defined by

$$z(\tau) = \int_0^\tau v_{\text{ins}}(\tau')\, d\tau', \tag{7.22}$$

and that from depth to time by

$$\tau(z) = \int_0^z \frac{dz'}{v_{\text{ins}}(z')}, \tag{7.23}$$

where v_{ins} is the *instantaneous* velocity or, equivalently, the local wave speed.

Though not migrations, these one-dimensional operations are very useful for converting from migrated depth to migrated time (or the reverse) or from unmigrated time to unmigrated depth (or the reverse). Thus it is possible to have time or depth displays both before and after migration, and the interpreter must be careful to understand what is being presented.

A form of aliasing is possible in all such operations (including migrations), though it is most easily analyzed in one dimension. Consider the conversion of a trace $s(t)$ to a depth trace $\hat{s}(z)$ with a desired depth sample interval of Δz. The depth trace will usually be constructed by a loop over the desired depth samples. Assuming constant velocity for simplicity, a particular depth sample of $\hat{s}(z)$ at $z = n\,\Delta z$ must be interpolated from $s(t)$ at $t = 2n\,\Delta z/v$. Thus, the time-to-depth conversion will effectively extract samples from $s(t)$ at the interval $\Delta t_{\text{eff}} = 2\,\Delta z/v$. This is a form of resampling. If $\Delta t_{\text{eff}} > \Delta t$, then aliasing will occur as the depth trace is constructed unless an antialias filter is applied. If $f_{\max}$ is the maximum signal frequency (presumed to be less than the Nyquist frequency) in $s(t)$, then the *sampling theorem* (Karl, 1989) ensures that $s(t)$ can be resampled to $\Delta t \le 1/(2f_{\max})$ (with antialias filtering to prevent noise aliasing) without loss of signal. Therefore, the depth sample interval should not be chosen arbitrarily but should be chosen to ensure that Δt_{eff} preserves signal. This leads to the condition

$$\Delta z \le \frac{v_{\min}}{4f_{\max}}. \tag{7.24}$$

In this equation, $v_{\min}$ is used to ensure that all signal frequencies are preserved in the worst case.

These considerations are true for multidimensional migrations as well as for simple stretching. The depth sample interval should be chosen with knowledge of the maximum signal frequency. Furthermore, the temporal data should be high-cut filtered to reject frequencies larger than the maximum frequency used to compute the depth sample interval.

7.2.2 Raytrace Migration of Normal-Incidence Seismograms

Most modern migration techniques are based on wavefield concepts that treat the recorded data as a boundary value. This was not always the case, however, because raytrace migration was very popular before 1980. As is often true, raytrace methods provide a great deal of insight into the problem and can be adapted to almost any setting. Understanding raytrace migration can help considerably in understanding the corresponding wavefield technique. For example, the very word "migration" comes from a popular, though somewhat incorrect, view that the process just "moves events around" on a seismic section. This encourages a fundamental, and very pervasive, confusion that time and depth sections are somehow equivalent. A better view is that an unmigrated time section and a migrated depth section are independent (in fact orthogonal) representations of a wavefield. Raytrace migration emphasizes this latter view and makes the role of the velocity model very clear.

Figure 7.6 illustrates the link between reflector dip and the measurable traveltime gradient (for normal-incidence rays) on a ZOS. The figure shows two rays that have normal incidence on a dipping reflector beneath a $v(z)$ medium. More general media are also easily handled; this just simplifies the discussion. The two rays are identical except for an extra segment in the lower ray in the bottom layer. From the geometry, this extra segment

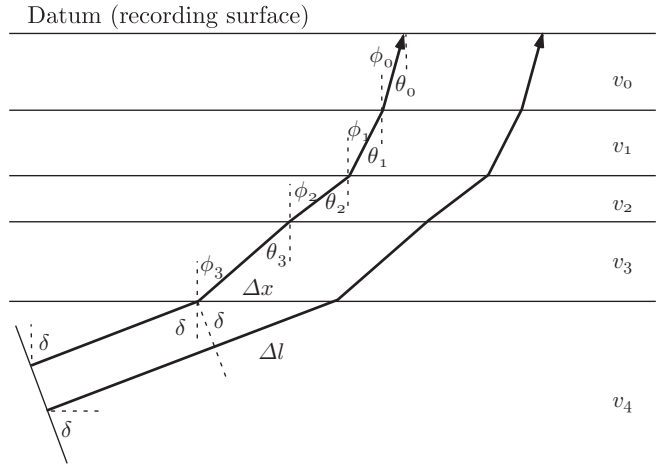


Figure 7.6 Two normal-incidence rays from a dipping reflector beneath a $v(z)$ medium. The delay between the rays allows the ray parameter to be estimated.

has a length $\Delta x \sin\delta$, where Δx is the horizontal distance between the rays. If there were receivers on the interface between layers 3 and 4, then for the upcoming rays, they would measure a traveltime delay, from left to right, of

$$\Delta t = \frac{2\,\Delta x \sin\delta}{v_4}, \tag{7.25}$$

where the factor of 2 accounts for two-way travel. This implies a horizontal traveltime gradient (horizontal slowness) of

$$\frac{\Delta t}{\Delta x} = 2\frac{\sin\delta}{v_4}. \tag{7.26}$$

Inspection of the geometry of Figure 7.6 shows that Eq. (7.26) can be rewritten as

$$\frac{\Delta t}{\Delta x} = 2p_{\mathrm{n}}, \tag{7.27}$$

where p_{n} is the ray parameter of the normal-incidence ray. In the $v(z)$ medium above the reflector, Snell's law ensures that p_{n} will remain equal to twice the horizontal slowness at all interfaces, including the surface at $z = 0$. Thus, a measurement of $\Delta t/\Delta x$ at the surface of a *normal-incidence seismogram* completely specifies the raypaths provided that the velocity model is also known. In fact, using Eq. (7.26) allows an immediate calculation of the reflector's dip.

Of course, in addition to the reflector dip, the spatial positions of the reflection points of the normal rays are also required. Also, it must be expected that more general velocity distributions than $v(z)$ will be encountered. In $v(x, y, z)$ media, the link between reflector dip and horizontal slowness is not as direct, but knowledge of one still determines the other. *Normal-incidence raytrace migration* provides a general algorithm that handles these complications. This migration algorithm will be described here for 2D but is easily generalized to 3D. Before presenting the algorithm, a formal definition of a *pick* is helpful:

Pick A *pick* is defined to be a triplet of values $(x_0, t_0, \Delta t/\Delta x)$ measured from a ZOS having the following meaning:

- x: inline coordinate at which the measurement is made.
- t_0: normal-incidence traveltime (two-way) measured at x.
- $\Delta t/\Delta x$: horizontal gradient of normal-incidence traveltime measured at (x, t_0).

To perform a raytrace migration, the following materials are required:

- $(x_i, t_{0ij}, \Delta t/\Delta x_{ij})$: a set of picks to be migrated. The picks are made at the locations x_i and at each location a number of t_{0ij} and $\Delta t/\Delta x_{ij}$ values are measured.
- $v(x, z)$: a velocity model is required. It must be defined for the entire range of coordinates that will be encountered by the rays.
- *Computation aids*: one or more of a calculator, compass, protractor, computer.

Finally, the algorithm is presented as a series of steps with reference to Figure 7.7. For all locations $x_i, i = 1, 2, \ldots, n$, each ray must be taken through the following steps:

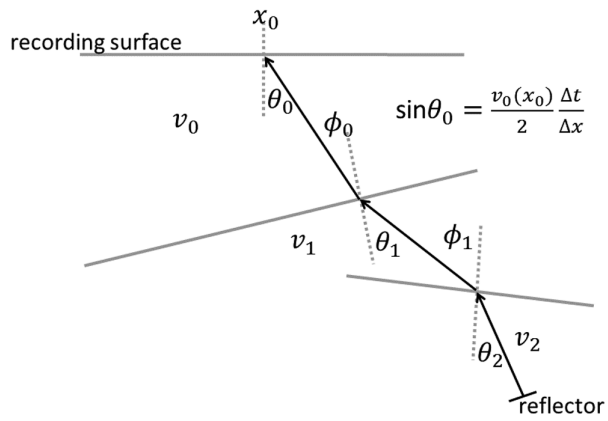


Figure 7.7 The migration of a single normal-incidence ray is shown for the case of three layers with two velocity interfaces. Knowledge of the traveltime gradient (time dip) allows inference of the emergence angle, θ_0, and knowledge of the velocity model then allows the ray to be traced down to the reflection point. At each interface the ray refracts according to Snell's law, and the ray terminates when the two-way traveltime along the ray equals the measured traveltime, t_0. The reflector is then inferred to be at a right angle to the raypath at its termination point.

Step 1. Determine the emergence angle at the surface of the *ij*th ray according to the formula

$$\sin\theta_{0ij} = \frac{v_0(x_i)}{2}\frac{\Delta t}{\Delta x_{ij}}. \tag{7.28}$$

Step 2. Denote the current layer by the number h. Project the ray down to the next interface and determine the incident angle ϕ_{hij}.

Step 3. Determine the length of the *ij*th raypath in the current (hth) layer, l_{hij}, and compute the layer traveltime

$$\delta t_{hij} = \frac{l_{hij}}{v_h}. \tag{7.29}$$

Step 4. Determine the refraction angle into the $(h+1)$th layer from Snell's law:

$$\sin(\theta_{(h+1)ij}) = \frac{v_{h+1}}{v_h}\sin(\phi_{hij}). \tag{7.30}$$

Step 5. Repeat steps 2–4 for each velocity layer until the stopping condition

$$t_{ij} = \sum_{h=1}^{n}\delta t_{hij} = \frac{t_{0ij}}{2} \tag{7.31}$$

is fulfilled, which simply says that the raypath traveltime must be half the measured traveltime.

Step 6. Draw in a reflecting segment perpendicular to the raypath at the point at which the stopping condition is satisfied.

7.2.3 Time and Depth Migration via Ray Tracing

When the first wavefield migration methods were being developed, the finite-difference approach was pioneered by Jon Claerbout (Claerbout, 1976) and his students at Stanford University. This very innovative work produced dramatic improvements in the quality of seismic images but the algorithms were a strong challenge for the available computers. As a result, approximations were sought which could reduce the numerical effort required. A key approximation was Claerbout's use of a moving coordinate system to derive an approximate wave equation that essentially substituted vertical traveltime, τ, for depth. When the finite-difference machinery was implemented with this approximate equation the resulting migrated images had a vertical coordinate of time, not depth. It also resulted, for reasons to be discussed when finite-difference methods are presented, in much faster run times. One way to see why this might be is to imagine a stacked section consisting only of events with $\Delta t/\Delta x = 0$. This is not far from the appearance of seismic sections from many of the world's sedimentary basins. Such basins are good $v(z)$ environments and the normal rays for flat events are all vertical. Thus, they are already positioned correctly in vertical traveltime on the stacked section and the migration algorithm has very little to do. This is a drastic oversimplification that will be more correctly stated later, but it comes close to the truth.

These early finite-difference methods were used with great success in sedimentary basins around the world. However, when the technology was taken into thrust belts, continental margins, and other areas where the $v(z)$ approximation is not a good model, it was gradually realized that systematic imaging errors were occurring. Today, the reason is well understood and it is a consequence of the approximate wave equation discussed in the previous paragraph. Figure 7.8 suggests the problem. Here an anticline is shown positioned beneath a slower-velocity near-surface region that has a dipping bottom interface. The normal ray from the crest of the structure is generally not the ray of minimum traveltime, even though it has the shortest path to the surface. Instead, a ray from a point to the right is the least-time ray because it spends more of its path in the fast material. Thus, on a normal-incidence section, the traveltime signature of the anticline will have a crest at the emergence point of the least-time ray. Generally, a traveltime crest will have zero $\Delta t/\Delta x$, which means that the corresponding ray emerges vertically. It turned out that the migration schemes were producing a distorted image in cases like this and the distortion placed the crest of the migrated anticline at the emergence point of the least-time ray. Just like in the sedimentary basin case, the algorithm was leaving flat events alone even though, in this case, it should not.

The understanding and resolution of this problem resulted in the terms *time migration* and *depth migration*. The former refers to any method that has a bias toward flat events and that works correctly in $v(z)$ settings. The latter refers to newer methods that were developed to overcome these problems and therefore can produce correct images even with strong lateral velocity gradients. Robinson (1983) presented the raytrace migration analogs to both time and depth migration, and these provide a great deal of insight. The raytrace migration algorithm presented previously in Section 7.2.2 is a depth migration algorithm because it obeys Snell's law even when $\partial_x v$ is significant.

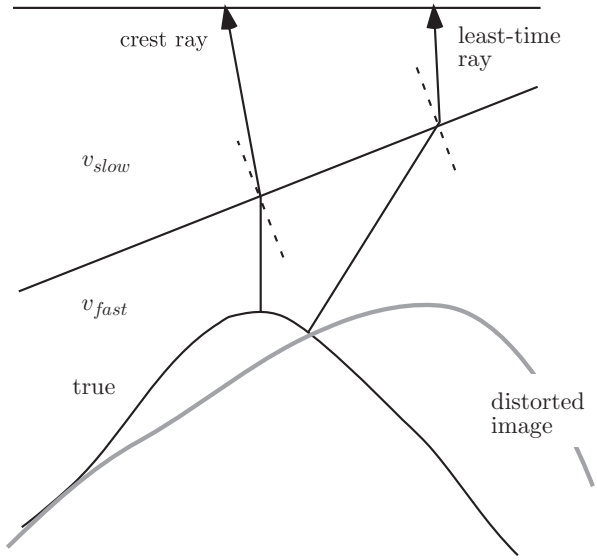


Figure 7.8 The least-time ray does not always come from the crest of an anticline.

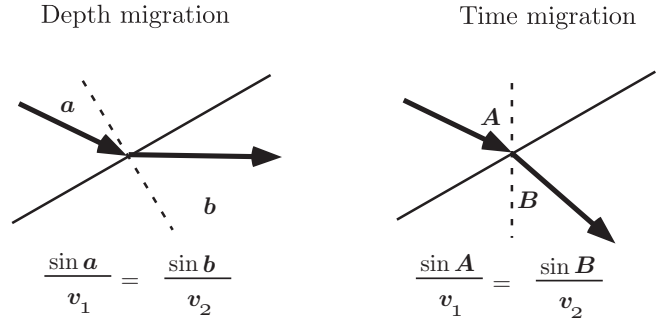


Figure 7.9 (left) When a ray encounters a dipping velocity interface, Snell's law predicts the transmitted ray using a relation between angles with respect to the interface normal. (right) In a time migration algorithm, Snell's law is systematically violated because the vertical angles are used. Both techniques give the same result if the velocity interface is horizontal.

Figure 7.9 shows how Snell's law must be systematically altered if a time migration is desired. Instead of using the angles that the incident and transmitted rays make with respect to the normal in Snell's law, the time migration algorithm uses the angles that the rays make with respect to the vertical. It is as though the dipping interface is rotated locally to the horizontal at the incident point of the ray. Thus, any vertical ray (i.e., with $\Delta t/\Delta x = 0$) will pass through the velocity interface without deflection, regardless of the magnitude of the velocity contrast.

This explains why time migration gets the wrong answer, and it also explains why the technology remains popular despite this now well-understood shortcoming. The fact that flat events are unaffected by time migration regardless of the velocity model means that the process has a "forgiving" behavior when the velocity model has errors. Velocity models derived automatically from stacking velocities tend to be full of inaccuracies. When used with time migration, these inaccuracies have little effect, but when used with depth migration they can be disastrous. Even today, the majority of seismic sections come from $v(z)$ settings and consist mostly of flat events. Such data can be migrated with relatively little care using time migration, and well-focused sections are obtained. Of course, these are still time sections, and a pleasing time migration does not mean that the migration velocities can be used for depth conversion. In general, the velocity errors that are "ignored" by time migration will cause major problems in a subsequent depth conversion. Much more care must be taken if an accurate depth section is required.

The terms *time migration* and *depth migration* arose in part because the early versions of both technologies tended to produce only time and depth sections, respectively. However, this is no longer the case, and either technology can produce both time and depth sections. Therefore, the terms should be treated simply as jargon that indicates whether or not an algorithm is capable of producing a correct image in the presence of strong lateral velocity gradients. The mere presence of a time or depth axis on a migrated seismic display says nothing about which technology was used. In this book, time migration will be used to refer to any migration technique that is strictly valid only for $v(z)$, while depth migration will refer to a technique that is valid for $v(x, z)$. Thus, when employed in constant-velocity or $v(z)$ settings, both techniques are valid and there is no meaningful distinction. Under these conditions, a migrated time section can always be converted to a depth section (or vice versa) with complete fidelity.

The first time migration algorithms were almost always finite-difference methods. Today, a better understanding of the subject allows virtually any migration method to be recast as a time migration. Therefore, it is very important to have some knowledge of the algorithm, or to have a statement from the developer, to be sure of the nature of a method. The issue has been clouded even further by the emergence of intermediate techniques. Sometimes it is worth the effort to build a synthetic seismic section to test a migration algorithm whose abilities are uncertain. One thing remains clear: depth migrations always require much more effort for a successful result. Not only is more computer time required, but also, much more significantly, vastly greater human time may be necessary to build the velocity model.

7.2.4 Elementary Wavefront Techniques

Though very versatile, the raytrace method described in Section 7.2.2 has its limitations. For example, it is not obvious how to make the migrated amplitudes any different from the unmigrated ones, and it is equally obvious that amplitudes should change under migration. Recall that band-limited reflectivity is the goal and that the geometrical spreading of wavefronts must be accounted for to estimate this (see Section 7.1.1). Wavefront methods

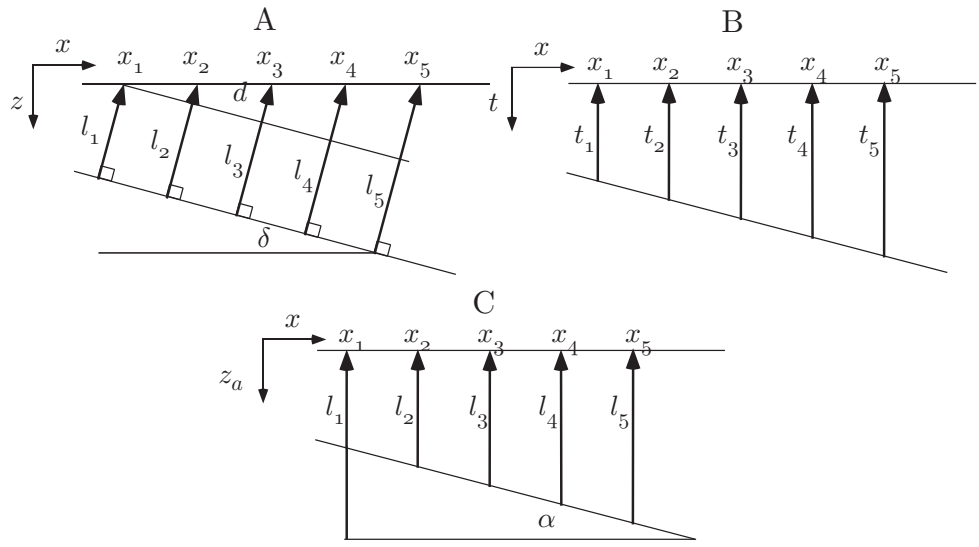


Figure 7.10 Depth, time, and apparent depth. (A) Normal-incidence raypaths for a dipping reflector in a constant-velocity medium. (B) The traveltime section for panel A plots the arrival time for each raypath beneath its emergence point. (C) When panel B is plotted at the apparent depth $z_a = vt/2$, an apparent dip can be defined.

provide amplitude adjustments in a very natural way, though they are more difficult to adapt to complex media than the raytrace approach is.

Hagedoorn (1954) provided one of the first, and still very relevant, explanations of wavefront migration. The ideas presented in this section are all essentially attributable to him. Consider the ZOS image of a dipping reflector overlaid by a constant-velocity medium as shown in Figure 7.10A. The reflector dip, δ, can be related to the lengths of the raypaths and their emergence points by

$$\sin\delta = \frac{l_5 - l_1}{x_5 - x_1}. \tag{7.32}$$

The time section corresponding to this raypath diagram is shown in Figure 7.10B. Since the velocity is constant, the arrival times are directly proportional to the path lengths through $t_k = 2l_k/v$, and the ZOS image of the reflector is seen to have a time dip of

$$\frac{\Delta t}{\Delta x} = \frac{t_5 - t_1}{x_5 - x_1} = \frac{2}{v}\frac{l_5 - l_1}{x_5 - x_1} = 2\frac{\sin\delta}{v}, \tag{7.33}$$

which is in agreement with Eq. (7.27) that the time dip on a ZOS gives twice the normal-incidence ray parameter. A further conceptual step can be taken by mapping the traveltimes to an *apparent depth*, z_a, by $z_a = vt/2$ as shown in Figure 7.10C. At this stage, the unmigrated display has both coordinate axes in distance units so that it makes sense to define

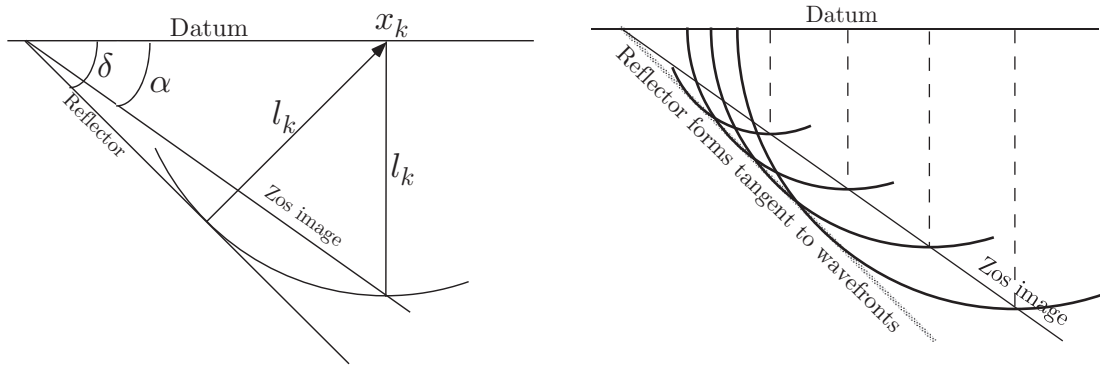


Figure 7.11a (left) The normal-incidence reflection point and the corresponding point on the ZOS image both lie on a circle of radius l_k centered at x_k.

Figure 7.11b (right) The wavefront migration of a dipping reflector. Each point on the ZOS image is replaced by a wavefront circle of radius l_k. The wavefronts interfere constructively at the location of the actual reflector.

an *apparent dip* through

$$\tan\alpha = \frac{l_5 - l_1}{x_5 - x_1}. \tag{7.34}$$

Finally, comparison of Eqs. (7.32) and (7.34) gives the relation

$$\sin\delta = \tan\alpha. \tag{7.35}$$

Equation (7.35) is called the *migrator's equation* and is of conceptual value because it illustrates the geometric link between unmigrated and migrated dips. However, it should not be taken too far. It is not generally valid for nonconstant velocity and cannot be applied to a time section. Though this last point may seem obvious, it is often overlooked. It is not meaningful to talk about a dip in degrees for an event measured on a time section, because the axes have different units. A change of timescale changes the apparent dip. On an apparent-depth section, an apparent dip can be meaningfully defined and related to the true dip through Eq. (7.35). The apparent dip can never be larger than 45°, because $\sin\delta \leq 1$, which is a restatement of the fact that the maximum time dip is $\Delta t/\Delta x = 1/v$ as discussed in Section 6.11.1. Though intuitively useful, the apparent-dip concept is not easily extended to variable velocity. In contrast, the limitation on maximum time dip can be simply restated for $v(z)$ media as $\Delta t/\Delta x = 1/v_{\min}$, where $v_{\min}$ is the minimum of $v(z)$.

Comparison of Figures 7.10a and 7.10c shows that, for a given surface location x_k, the normal-incidence reflection point and the corresponding point on the ZOS image both lie on a circle of radius l_k centered at x_k. This relationship is shown in Figure 7.11a, from which it is also apparent that:

- The ZOS image lies at the nadir (lowest point) of the circle.
- The reflection point is tangent to the circle.
- The ZOS image and the reflector coincide on the datum (recording plane).

These observations form the basis of a simple wavefront migration technique which is kinematically exact for constant velocity:

1. Stretch the ZOS image from t to z_a (1D time–depth conversion).
2. Replace each point in the (x, z_a) picture with a wavefront circle of radius z_a. Amplitudes along the wavefront circle are constant and equal to the amplitude at (x, z_a).
3. The superposition of all such wavefronts will form the desired depth section.

Figure 7.11b shows the migration of a dipping reflector by wavefront superposition. The actual mechanics of the construction are very similar to a convolution. Each point in the ZOS image is used to scale a wavefront circle, which then replaces the point. A 2D convolution can be constructed by a similar process of replacement. The difference is that the replacement curve in a 2D convolution is the same for all points, while in the migration procedure it varies with depth. Thus the migration process can be viewed as a *nonstationary* convolution.

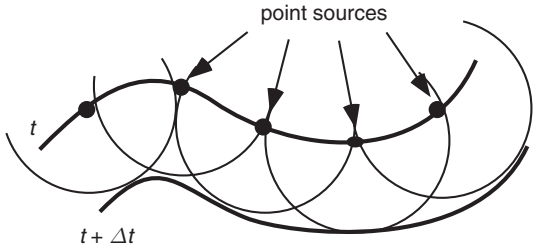
Wavefront migration shows that the *impulse response* of migration is a wavefront circle. That is, if the input ZOS contains only a single nonzero point, then the migrated section will be a wavefront circle. This circle is the locus of all points in (x, z) that have the same traveltime to $(x_k, 0)$. Two equivalent interpretations of this are either that the circle is the only Earth model that could produce a single output point or that the circle is the curve of *equal probability*. The first interpretation is a deterministic statement that only one Earth model could produce a ZOS image with a single live sample. The second interpretation is a stochastic one that suggests why the generalization from one live sample to many works. By replacing each point with its curve of equal probability, the migrated section emerges as the most likely geology for the superposition of all points. The constructive interference of a great many wavefronts forms the migrated image.

Exercises

7.2.1 Describe the appearance of:
- a migrated section that contains a high-amplitude noise burst on a single trace;
- a migration that results from a ZOS image whose noise level increases with time;
- the edge of a migrated section.

7.2.5 Huygens' Principle and Point Diffractors

Christiaan Huygens was an early physicist and astronomer (a contemporary of Newton) who made a number of advances in the understanding of waves. Most notable was his principle that, if the position of a wavefront at time t is known, its position at $t + \Delta t$ can be computed by a simple strategy. Each point on the wavefront at time t is considered to be a secondary source of a small spherical (3D) or circular (2D) wavefront called a *Huygens' wavelet*, as shown in Figure 7.12a. For the seismic problem, Huygens' principle



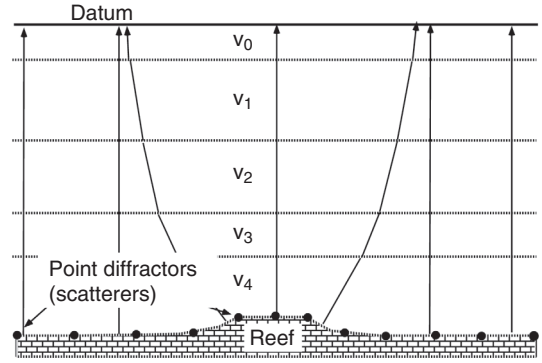
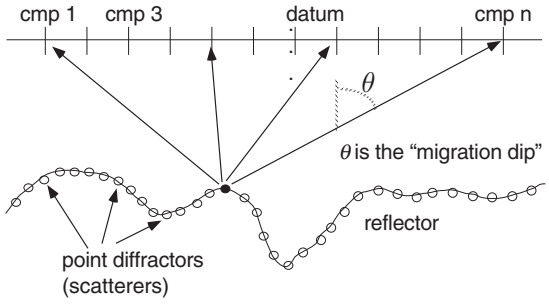


Figure 7.12a (left) Huygens' principle reconstructs the wavefront at $t + \Delta t$ by the superposition of small wavefronts from secondary point sources placed on the wavefront at t.

Figure 7.12b (right) The seismic response of a continuous geological structure is synthesized as the superposition of the responses of many *point diffractors*.



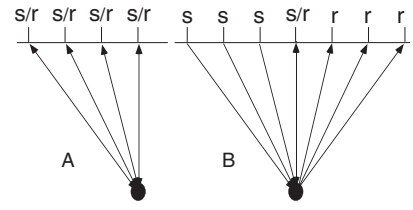


Figure 7.13a (left) Each point on a reflector is considered to be a point diffractor that scatters energy to all surface locations.

Figure 7.13b (right) (A) The raypaths for the left side of a zero-offset point diffractor. (B) The raypaths for a CMP gather assuming a zero-dip reflector. These raypaths are identical for $v(z)$, leading to the conclusion that the traveltime curves for the point diffractor are the same as for the NMO experiment.

can be adapted to consider the response of a continuous reflector as the superposition of the responses of a great many *point diffractors* or *scatterpoints* (Figure 7.12b).

Figure 7.13a shows the concept of a point diffractor that scatters any incident wave in all directions. Thus the complete imaging of any reflector is only possible if all of this scattered energy is captured and focused at the scatterpoint. Each scattered raypath is characterized by its scattering angle, θ. If the scatterpoint lies on a dipping reflector, then the scattered ray that coincides with the normal ray will be the strongest one after superposition of many scatterpoints. For this ray, the scattering angle is the same as the reflector dip, which suggests why the scattering angle is commonly called the "migration dip" in a migration program. However, *migration dip* still conveys a misleading impression to many untutored users, who wish to equate it with geologic dip. They draw the mistaken conclusion that, for data with low geologic dips, it suffices to correctly handle only low

migration dips. The correct interpretation of migration dip as *scattering angle* shows that even for low geologic dips it is desirable to handle large scattering angles. In fact, a high-resolution migration requires that the scattering angle be as large as possible. In the context of Section 7.1.3, the scattering angle is directly related to spectral components through $k_x/f = 0.5 \sin\theta/v$ or, equivalently, $k_x = 0.5f \sin\theta$. This says that high k_x values require large values of θ. It is a fundamental component of resolution theory that the k_x spectrum must be as large as possible for high resolution.

The traveltime curve of a point diffractor, for zero-offset recording and constant velocity, has a raypath geometry that is essentially equivalent to that for a CMP gather and a zero-dip reflector (Figure 7.13b). Recall that the NMO traveltime curve is given by $t_H^2 = t_0^2 + H^2/v^2$, where H is the full source–receiver offset. This means that the traveltime curve for a zero-offset point diffractor is given by

$$t_x^2 = t_0^2 + \frac{4(x - x_0)^2}{v^2}, \tag{7.36}$$

where the diffractor is at (x_0, z), $t_0 = 2z/v$, and the velocity, v, is constant. The factor of 4 in the numerator arises because $x - x_0$ corresponds to the half-offset. For $v(z)$, the conclusion is immediate that the Dix equation proof of the NMO hyperbola is approximately characterized by v_{rms} means that the diffraction traveltimes are approximately

$$t_x^2 \approx t_0^2 + \frac{4(x - x_0)^2}{v_{\text{rms}}^2}, \tag{7.37}$$

where $t_0 = 2z/v_{\text{ave}}$.

Figure 7.11b shows how wavefront superposition migrates the ZOS image of a dipping reflector. The ZOS image of a single point diffractor is the hyperbola given by Eq. (7.36), so it is instructive to see how wavefront migration can collapse such hyperbolas. Figure 7.14A shows a *diffraction chart* that gives the traveltime curves for five point diffractors in a constant-velocity medium. The wavefront migration of this chart should convert it into five impulses, one at the apex of each hyperbola. Figure 7.14B shows the wavefront migration of the second hyperbola on the chart. Each point on the second hyperbola has been replaced by a wavefront circle whose radius is the vertical time of the point. Since the chart is drawn with a vertical coordinate of time rather than depth, these wavefront curves may not appear to be exactly circular. The geometry of hyperbolas and circles is such that the wavefront circles all pass through the apex of the second hyperbola. Thus the amplitude of the wavefront superposition will be large at the apex and small elsewhere. The same process will also focus all other hyperbolas on the chart simultaneously.

The diffraction curves can be used to perform the inverse of migration, or *modeling*. Figure 7.15 shows the construction of the ZOS image of a dipping reflector using Huygens' principle. Each point on the dipping reflector is replaced by the ZOS image of a point diffractor. The superposition of these many hyperbolas constructs the ZOS image of the dipping reflector. For a given hyperbola, its apex lies on the geology and it is tangent to the ZOS image. This directly illustrates that the wavefront migration of dipping reflectors

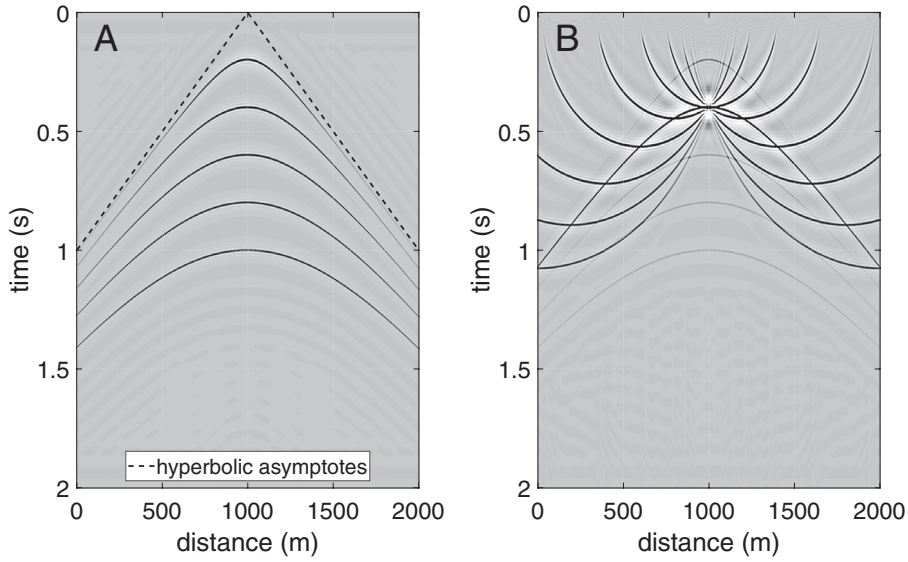


Figure 7.14 (A) This *diffraction chart* is actually a seismic matrix and shows the diffractions (traveltime curves) for five point diffractors in a constant velocity ($v = 2000$ m/s) medium. All five curves are asymptotic to the lines shown. (B) The second diffraction hyperbola has been amplified and migrated by wavefront superposition. Only a few sparse wavefronts are shown to illustrate the focusing effect. The migration was accomplished by the command *wavefrontmig*. These figures were created by the script `diffchart_mig.m`.

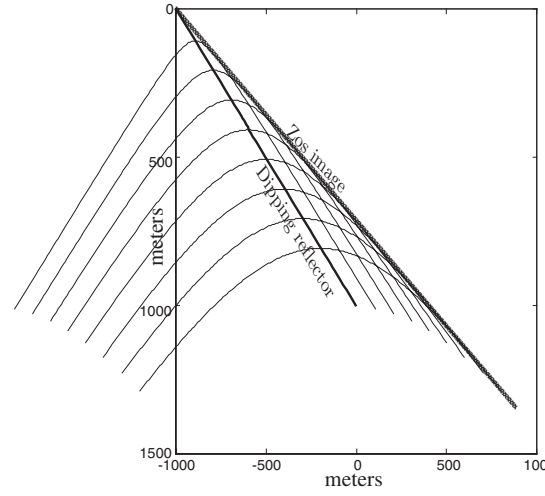


Figure 7.15 The ZOS image of a dipping reflector is formed by replacing each point on the reflector with the ZOS image of a point diffractor. The superposition of these many hyperbolas forms the dipping reflector's ZOS image.

is completely equivalent to the migration of diffraction responses. In fact, the collapse of diffraction responses is a complete statement of the migration problem, and a given algorithm can be tested by how well it achieves this goal.

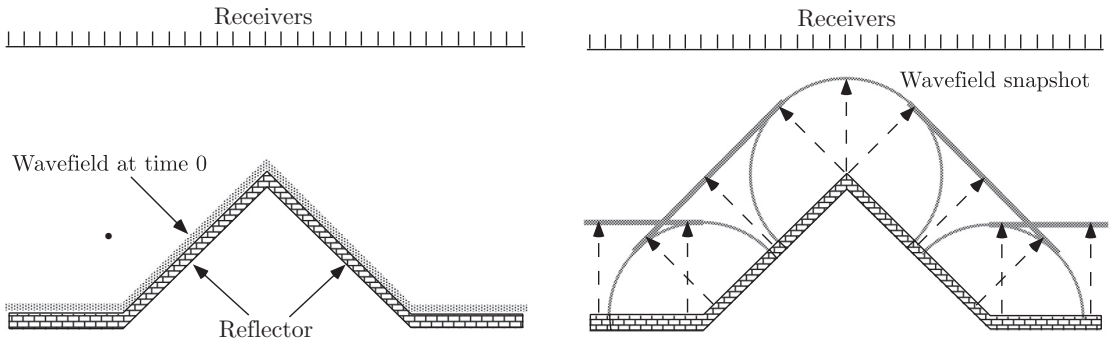


Figure 7.16a (left) The exploding reflector model at the instant of explosion establishes a wavefield that is isomorphic with the reflector.

Figure 7.16b (right) A snapshot of the ERM wavefield at some instant of time as it approaches the receivers.

7.2.6 The Exploding Reflector Model

In the preceding sections, two methods to migrate stacked data have been described. In Section 7.2.2, the stacked section was treated as a normal-incidence seismogram and ray-trace techniques were developed to migrate it. In Section 7.2.4, the wavefront migration method was presented for constant-velocity zero-offset sections. Further progress is greatly facilitated by a more abstract model of the stacked section. Ultimately, the goal is to formulate the poststack migration problem as a solution to the wave equation where the measured data plays the role of a *boundary value*. However, this is complicated by the fundamental fact that the CMP stack is not a single wavefield but a composite of many individual wavefields. There is no single physical experiment that could record a ZOS, and so a ZOS cannot be a single physical wavefield.

There is a useful thought experiment, called the *exploding reflector model* (ERM), that does yield something very similar to a ZOS and serves as the basis for most poststack migration methods. (The following discussion is presented in 2D, and the generalization to 3D is elementary.) As motivation, note that Eq. (7.36) can be rewritten as

$$t_x^2 = t_0^2 + \frac{(x - x_0)^2}{\hat{v}^2}, \tag{7.38}$$

where $\hat{v} = v/2$ is called the *exploding reflector velocity* and $t_0 = z/\hat{v}$. This trivial recasting allows the interpretation that the point diffractor is a seismic source of waves that travel at one half of the physical velocity. As shown in Figure 7.16a, the exploding reflector model adopts this view and postulates a model identical to the real Earth in all respects except that the reflectors are primed with explosives and the seismic wave speeds are all halved. Receivers are placed at the CMP locations, and at $t = 0$ the explosives are detonated. This creates a wavefield that is morphologically identical to the geology at the instant of explosion.

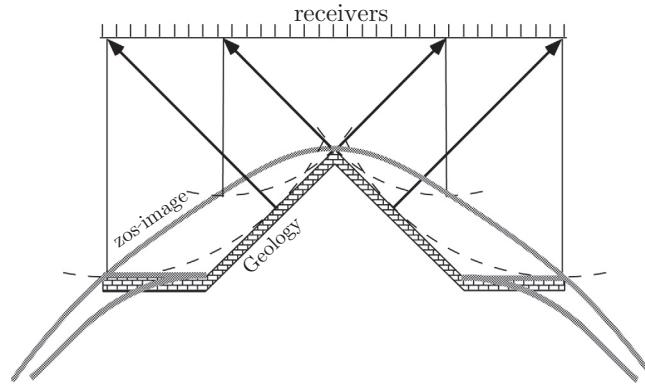


Figure 7.17 The ERM seismogram superimposed on the migrated depth section.

If $\psi(x, z, t)$ denotes the exploding reflector wavefield, then the mathematical goal of the migration problem is to calculate $\psi(x, z, t = 0)$. Thus $\psi(x, z, t = 0)$ is identified with the geology and represents the migrated depth section. The ERM wavefield is allowed to propagate only upward (the $-z$ direction), without reflections, mode conversions, or multiples. It refracts according to Snell's law and the half-velocities mean that the traveltimes measured at the receivers are identical to those of the ZOS. In the constant-velocity simulation of Figure 7.16b, a *snapshot* of the ERM wavefield, $\psi(x, z, t = \text{const})$, is shown as the wavefield approaches the receivers. The raypaths are normal-incidence raypaths (because of the isomorphism between the geology and $\psi(x, z, t = 0)$) and spreading and buried foci are manifest. The figure emphasizes three Huygens wavelets that emanate from the three corners of the geologic model. The migrated depth section is also a snapshot, but a very particular one, so the term *snapshot* will be used to denote all such constant-time views.

In Figure 7.17, the ERM seismogram, $\psi(x, z = 0, t)$, is shown in apparent depth superimposed on top of the depth section. Wavefront circles are shown connecting points on the geology with points on the seismogram. The ERM seismogram is kinematically identical to the normal-incidence seismogram (i.e., the traveltimes are the same) and is thus a model of the CMP stack. It is also kinematically identical to all normal-incidence primary events on a ZOS image. This allows the migration problem to be formulated as a solution to the wave equation. Given $\psi(x, z = 0, t)$ as a boundary condition, a migration algorithm solves the wave equation for the entire ERM wavefield, $\psi(x, z, t)$, and then sets $t = 0$ to obtain the migrated depth section. This last step of setting $t = 0$ is sometimes called *imaging*, though this term has also come to refer to the broader context of migration in general. Precisely how the wavefield is evaluated to extract the geology is called an *imaging condition*. In the poststack case, the imaging condition is a simple evaluation at $t = 0$. Later, it will be seen that there are other imaging conditions for prestack migration.

Thus far, the ERM has given simple mathematical definitions to the migrated depth section, the recorded seismogram, and the wavefield snapshot. In addition, the

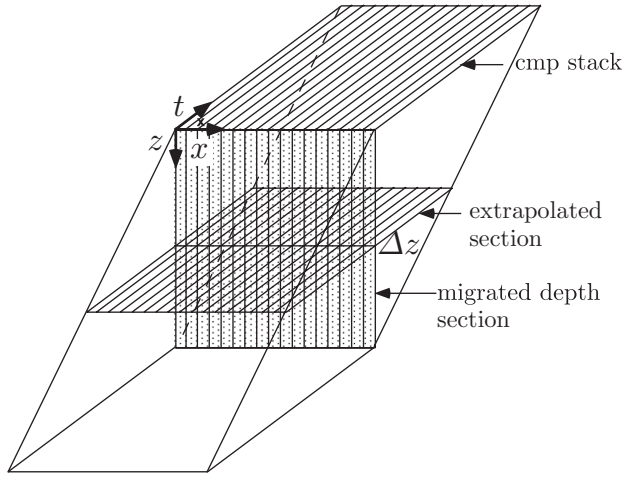


Figure 7.18 This prism shows the portion of (x, z, t) space that can be constructed from a finite-extent measurement of $\psi(x, z = 0, t)$ (upper face). Also shown are the migrated depth section, $\psi(x, z, t = 0)$ (vertical slice), and an extrapolated section, $\psi(x, z = \Delta z, t)$ (horizontal slice).

extrapolated seismogram can be defined as $\psi(x, z = \Delta z, t)$. Both the ERM seismogram and the extrapolated seismogram are time sections, while the snapshot and the migrated section are in depth. The extrapolated seismogram is a mathematical simulation of what would have been recorded had the receivers been at $z = \Delta z$ rather than at $z = 0$. The construction of $\psi(x, z = \Delta z, t)$ from $\psi(x, z = 0, t)$ is called *downward continuation* and, synonymously, wavefield extrapolation.

As a summary, the ERM defines the following quantities:

- $\psi(x, z, t)$: the ERM wavefield.
- $\psi(x, z, t = 0)$: the migrated depth section.
- $\psi(x, z, t = \text{const})$: a wavefield snapshot.
- $\psi(x, z = 0, t)$: the ERM seismogram.
- $\psi(z, z = \Delta z, t)$: an extrapolated section.

These quantities are depicted in Figure 7.18. It is significant that any extrapolated seismogram can be evaluated at $t = 0$ to give a single depth sample of the migrated depth section. The process of deducing a succession of extrapolated seismograms, and evaluating each at $t = 0$, is called a *recursive migration*. It is recursive because the extrapolated seismogram $\psi(x, z = z_k, t)$ is computed from the previous seismogram $\psi(x, z = z_{k-1}, t)$. Examples of recursive migrations are the finite-difference methods and the phase-shift techniques. An alternative to the recursive approach is a *direct migration* that computes $\psi(x, z, t = 0)$ directly from $\psi(x, z = 0, t)$ without the construction of any intermediate products. Examples of direct migration are *f–k* migration and Kirchhoff migration.

A direct migration tends to be computationally more efficient than a recursive approach in terms of both memory usage and computation speed. However, the direct methods

must deal with the entire complexity of the velocity structure all at once. In contrast, the recursive approach forms a natural partitioning of the migration process into a series of wavefield extrapolations. The extrapolation from $z = z_{k-1}$ to $z = z_k$ need only deal with the velocity complexities between these two depths. If the interval $z_{k-1} \rightarrow z_k$ is taken sufficiently small, then the vertical variation of velocity can be ignored and v can be considered to depend only upon the lateral coordinates. In this way, the migration for a complex $v(x, z)$ variation can be built from the solution of many $v(x)$ problems.

7.3 MATLAB Facilities for Simple Modeling and Raytrace Migration

This section describes three facilities that can produce simple seismic section models. The first two, hyperbolic superposition and finite differencing, create full waveform models suitable for migration with full waveform migration methods. The third, normal-incidence ray tracing, merely predicts arrival times of events and not their amplitudes.

7.3.1 Modeling by Hyperbolic Superposition

Section 7.2.5 describes how the seismic response of a complex geological structure can be viewed as the superposition of the responses of many point diffractors. Conceptually, infinitely many point diffractors are required, and the seismic response is found in the limit as the number of diffractors becomes infinite and their spacing infinitesimal. In a practical computer implementation, only a finite number of diffractors can be simulated and their spacing can be no finer than the underlying computation grid.

There are six basic routines that support modeling by superposition of hyperbolas. These are collected in the *synsections* toolbox and assume that a constant-velocity, zero-offset synthetic is desired. The fundamental paradigm is that each routine inserts a single event into a matrix that represents the seismic section. These six basic commands are:

event_spike Inserts an isolated noise spike.

event_hyp Inserts a hyperbolic event.

event_dip Inserts a dipping (linear) event.

event_diph Builds a dipping event by superimposing hyperbolas.

event_diph2 Similar to *event_diph*, plus it allows control over the spacing of the hyperbolas.

event_pwlinh Superimposes hyperbolas along a piecewise linear track.

These all operate similarly and insert a spike, hyperbola, linear event, or piecewise linear event into a matrix. The matrix must be created externally and assigned vertical (time) and horizontal (distance) coordinates. Then the inserted event is described by its position in (x, t) or (x, z). Some of the commands require the specification of velocity, and this should be the physical velocity. Within the functions, the velocity is halved to produce an

Code Snippet 7.3.1 This example creates the synthetic zero-offset section shown in Figure 7.19a.

```
1   v=2000;dx=10;dt=.004;%basic model parameters
2   x1=0:dx:2000;%x axis
3   t1=0:dt:2;%t axis
4   seis1=zeros(length(t1),length(x1));%allocate seismic matrix
5   seis1=event_hyp(seis1,t1,x1,.4,700,v,1,3);%hyperbolic event
6   seis1=event_dip(seis1,t1,x1,[.75 1.23],[700 1500],1);%linear event
7   [w,tw]=ricker(dt,40,.2);%make ricker wavelet
8   seis1=sectconv(seis1,t1,w,tw);%apply wavelet
```

End Code

elmigcode/eventexample1.m

Code Snippet 7.3.2 This is similar to Code Snippet 7.3.1 but differs by using *event_diph* to create the dipping event. The result is shown in Figure 7.19b.

```
1   v=2000;dx=10;dt=.004;%basic model parameters
2   x1=0:dx:2000;%x axis
3   t1=0:dt:2;%t axis
4   seis1=zeros(length(t1),length(x1));%allocate seismic matrix
5   seis1=event_hyp(seis1,t1,x1,.4,700,v,1,3);%hyperbolic event
6   seis1=event_dip(seis1,t1,x1,[.75 1.23],[700 1500],1);%linear event
7   [w,tw]=ricker(dt,40,.2);%make ricker wavelet
8   seis1=sectconv(seis1,t1,w,tw);%apply wavelet
```

End Code

elmigcode/eventexample1.m

exploding reflector synthetic. The final three commands create events by the superposition of hyperbolas along linear or piecewise linear structures. By calling these functions repeatedly, the seismic responses from quite complex geometric shapes can be synthesized.

Code Snippet 7.3.1 illustrates the use of *event_hyp* and *event_dip* to create the model section shown in Figure 7.19a. The first three lines establish the basic model geometry and define the coordinate axes; then, line 4 initializes the seismic matrix to zero. On line 5, *event_hyp* is invoked to make the hyperbolic diffraction response shown in the top left of Figure 7.19a. The seismic matrix is input to *event_hyp* and is replaced by the output. The coordinate vectors (second and third input parameters) are required to define the geometry. The fourth and fifth input parameters are the (x, t) coordinates of the apex of the diffraction, and the sixth parameter is the velocity. The seventh parameter sets the amplitude at the apex of the hyperbola, and the eighth is a flag that determines how amplitude decays down the limbs of the hyperbola. There are four possibilities: (1) no amplitude decay, (2) a decay given by $a(x) = a(0)t_0/t_x$ (where $a(x)$ is the amplitude at offset x, t_0 is the zero-offset traveltime, and t_x is the traveltime at offset x), (3) a decay given by $a(x) = a(0)(t_0/t_x)^{3/2}$, and (4) a decay given by $a(x) = a(0)(t_0/t_x)^2$. Line 6 invokes *event_dip* to create the linear dipping event seen in the center of Figure 7.19a. The fourth input parameter specifies the

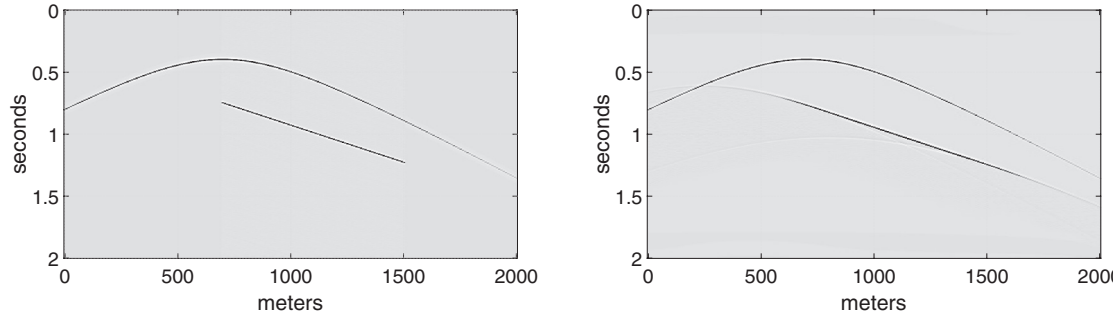


Figure 7.19a (left) The hyperbolic response of a point diffractor and a simple linear event. The display is slightly clipped to make the diffraction more visible. This was created by Code Snippet 7.3.1. The migration of this data is shown in Figure 7.32a.

Figure 7.19b (right) Similar to Figure 7.19a except that the linear event was created by the superposition of many hyperbolas. The display is clipped to match that of Figure 7.19a. See Code Snippet 7.3.2. The migration of this data is shown in Figure 7.32b.

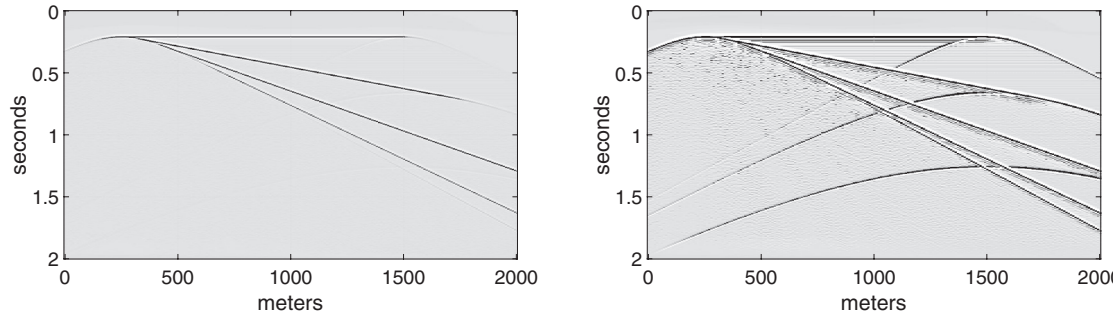


Figure 7.20a (left) The response of five reflectors with dips of 0°, 20°, 40°, 60°, and 80°. The migration of this data is shown in Figure 7.33a.

Figure 7.20b (right) The same dataset as in Figure 7.20b is shown with a strongly clipped display to reveal the underlying hyperbolas.

time of the event at its beginning and end, while the fifth parameter gives the corresponding lateral positions. The last parameter gives the event amplitude. Once the events are created, the final two lines make a Ricker wavelet (with a dominant frequency of 40 Hz) and convolve it with the seismic section.

The linear event shown in Figure 7.19a is physically unrealistic because it lacks diffractions from its endpoints. Real wavefields are never discontinuous. A much more realistic event can by created using `event_diph`. This function synthesizes a linear event by hyperbolic superposition exactly as illustrated in Figure 7.15. The result is the seismic section shown in Figure 7.19b. In order to do so, the input parameters must describe the dipping event in (x, z) rather than (x, t). Code Snippet 7.3.2 shows how this is done and differs from Code Snippet 7.3.1 only on line 6. The fourth input parameter for `event_diph`

Code Snippet 7.3.3 This code illustrates the use of *event_pwlinh* to create a simple model of a reef. The result is shown in Figure 7.21a.

```
v=2000;dx=10;dt=.004;%basic model parameters
x4=0:dx:3000;%x axis
t4=0:dt:1.5;%t axis
zreef=600;hwreef=200;hreef=100;%depth, half-width, and height of reef
xcntr=max(x4)/2;
xpoly=[xcntr-hwreef xcntr-.8*hwreef xcntr+.8*hwreef xcntr+hwreef];
zpoly=[zreef zreef-hreef zreef-hreef zreef];
seis4=zeros(length(t4),length(x4));%allocate seismic matrix
seis4=event_diph(seis4,t4,x4,v,0,xcntr-hwreef,zreef,0,.1);%left
%right
seis4=event_diph(seis4,t4,x4,v,xcntr+hwreef,max(x4),zreef,0,.1);
%base
seis4=event_diph(seis4,t4,x4,v,xcntr-hwreef,xcntr+hwreef,...
    zreef,0,.2);
%top
seis4=event_pwlinh(seis4,t4,x4,v,xpoly,zpoly,...
    -.1*ones(size(zpoly)));
[w,tw]=ricker(dt,40,.2);%make ricker wavelet
seis4=sectconv(seis4,t4,w,tw);%apply wavelet
```

End Code

elmigcode/eventexample4.m

is the velocity, while the next four parameters are, respectively, the starting and ending x coordinates, the starting z coordinate, and the dip in degrees. The lateral extent of the resulting event is generally much greater than that of the prescribed coordinates. If the section is migrated, then the migrated reflector will be found between the specified coordinates.

A more complex seismic section is shown in Figure 7.20a. The geologic model is a fan of five dipping reflectors that originate at the point $(x, z) = (250\,\text{m}, 200\,\text{m})$ and extend to the right until $x = 1500\,\text{m}$. In Figure 7.20b, the same seismic section is shown with a strongly clipped display to reveal the underlying hyperbolas.

Code Snippet 7.3.3 illustrates the use of these hyperbolic summation tools to create a simple model of a reef. The resulting seismic response is shown in Figures 7.21a and 7.21b. The reef is a simple trapezoidal structure, 400 m wide and 50 m high, on top of a flat reflector 600 m below the recording plane. The reflection coefficient of the reef is modeled as -0.1 (the acoustic impedance within the reef is assumed to be lower than that of the surrounding material), $+0.1$ on the base reflector, and $+0.2$ beneath the reef.

The function *event_diph2* is similar to *event_diph* except that it allows control over the spacing of hyperbolas through the input parameter `ndelx`. In *event_diph* the hyperbola spacing is never greater than the grid spacing, while in *event_diph2* the spacing is `ndelx` times greater than in *event_diph*. Code Snippet 7.3.4 illustrates the use of this function to create a series of figures illustrating the gradual formation of the seismic response of a trapezoidal structure. The sequence of Figures 7.22a, 7.22b, 7.23a, and

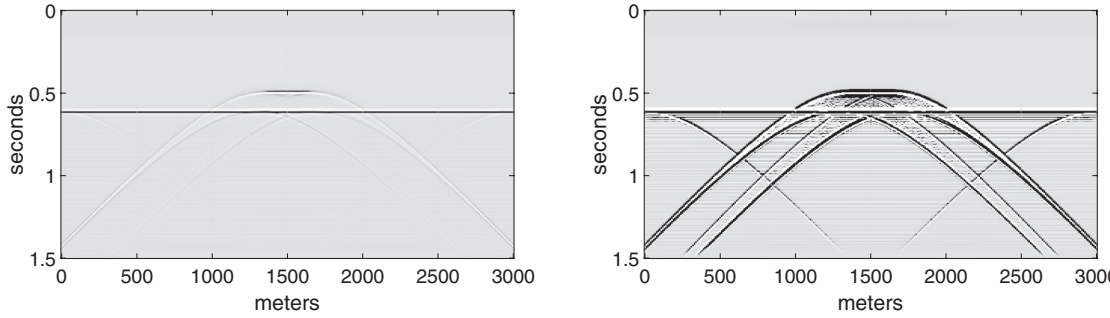


Figure 7.21a (left) The seismic response of a simple reef model. This was created with Code Snippet 7.3.3. The migration of this data is shown in Figure 7.33b.

Figure 7.21b (right) The same dataset as in Figure 7.21a is shown with a strongly clipped display to reveal the underlying hyperbolas.

Code Snippet 7.3.4 This code uses *event_diph2* to construct the response of a trapezoidal structure. The parameter `ndelx` (specified externally to this snippet) controls the spacing of the hyperbolas. Figures 7.22a, 7.22b, 7.23a, and 7.23b correspond to values of `ndelx` of 15, 10, 5, and 1, respectively.

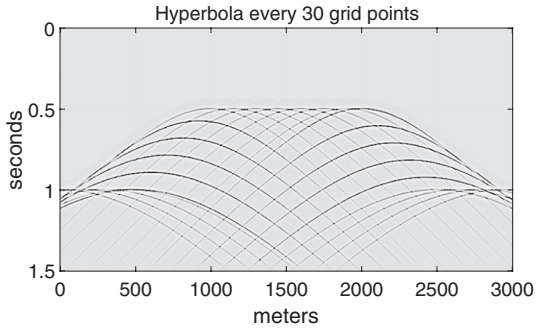
```
v=2000;dx=5;dt=.004;%basic model parameters
x5=0:dx:3000;%x axis
t5=0:dt:1.5;%t axis
xcntr=max(x5)/2;
seis5=zeros(length(t5),length(x5));%allocate seismic matrix
seis5=event_diph2(seis5,t5,x5,v,0,500,1000,ndelx,0,.1);
seis5=event_diph2(seis5,t5,x5,v,500,xcntr-500,1000,ndelx,-45,.1);
seis5=event_diph2(seis5,t5,x5,v,xcntr-500,xcntr+500,500,ndelx,...
    0,.1);
seis5=event_diph2(seis5,t5,x5,v,xcntr+500,max(x5)-500,500,...
    ndelx,45,.1);
seis5=event_diph2(seis5,t5,x5,v,max(x5)-500,max(x5),1000,...
    ndelx,0,.1);
[w,tw]=ricker(dt,40,.2);%make ricker wavelet
seis5=sectconv(seis5,t5,w,tw);%apply wavelet
```

End Code

elmigcode/eventexample5.m

7.23b shows the complete seismic response gradually forming as the density of hyperbolas increases.

Also present in the *synsections* toolbox are two functions that make "standard" synthetics called *makestdsyn* and *makestdsynh*. These differ in that one uses hyperbolic superposition for linear events and the other does not. Their use is illustrated in the script *makesections*, which can be run as a demonstration.



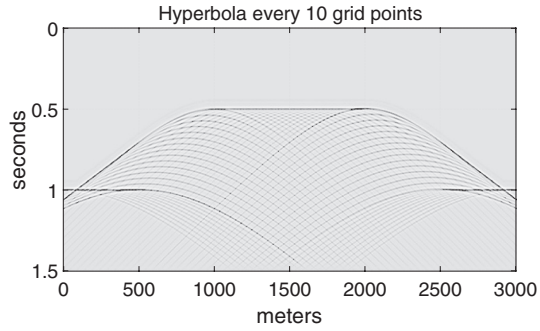
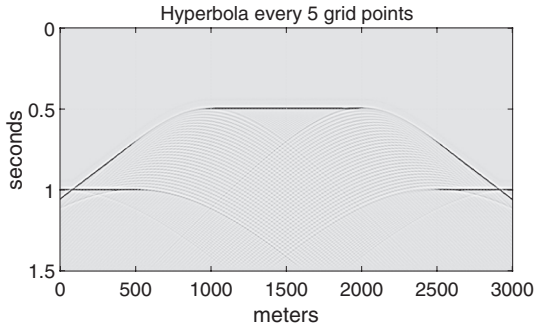


Figure 7.22a (left) A trapezoidal structure is modeled with only a few sparse hyperbolas. Relative to Figure 7.23b, only every thirtieth hyperbola is shown. The migration of this data is shown in Figure 7.34a.

Figure 7.22b (right) Similar to Figure 7.22a except that every tenth hyperbola is shown.



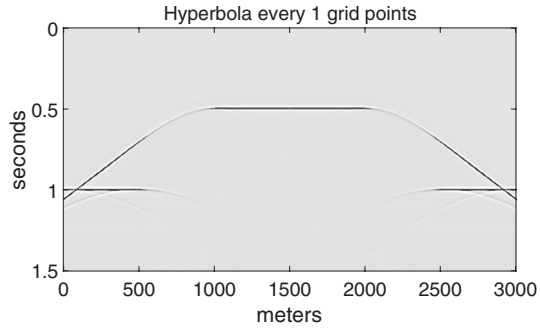


Figure 7.23a (left) Similar to Figure 7.22a except that every fifth hyperbola is shown.

Figure 7.23b (right) A trapezoidal structure is completely modeled by hyperbolic superposition. Compare with Figures 7.22a, 7.22b, and 7.23a. The migration of this data is shown in Figure 7.34b.

7.3.2 Finite-Difference Modeling for Exploding Reflectors

This section builds on Section 4.5 and describes those extensions required for exploding reflector modeling. The key facility here is the function `afd_explode`, which uses an input velocity model to calculate the initial exploding reflector snapshot, $\psi(x, z, t = 0)$, and then time step it with Eq. (4.58).

The calculation of $\psi(x, z, t = 0)$ can be done directly from the velocity model if it is assumed to be the normal-incidence reflectivity at constant density. Using Eq. (7.1) and assuming a unit volume, this is

$$\psi(x, z, t = 0) = r_{\mathrm{n}}(x, z) \approx \frac{1}{2} \frac{\partial \log v(x, z)}{\partial z}. \tag{7.39}$$

This assumes that the normal derivative can be approximated by the z derivative, which is reasonable for low-relief structures. This can be simply calculated using MATLAB's

built-in gradient function, `gradient`. Given an input matrix representing $\log v(x,z)$, this function returns two matrices of the same size representing $\partial_x \log v(x,z)$ and $\partial_z \log v(x,z)$. These are calculated using a centered finite-difference approximation for interior points and one-sided approximations on the boundaries. Thus, the reflectivity is calculated by

```
[rx,rz]=gradient(log(velocity));
r=.5*rz;
```

where `velocity` is a matrix representing $v(x,z)$. This calculation is provided by the function *afd_reflec*, which is invoked by *afd_explode*.

Code Snippet 7.3.5 illustrates the creation of a model of a small channel beneath a layered medium. The creation of the velocity model uses *afd_vmodel* and was discussed in detail in Section 4.5. The channel is defined on lines 25–29 as 20 m wide and 50 m deep. With the 5 m grid (line 1), this means that the channel is 4 samples wide and 10 samples deep. The resulting velocity model is shown in Figure 7.24a. Lines 32–36 take this velocity model into *afd_explode* to create an exploding reflector seismogram using the fourth-order Laplacian approximation (Eq. (4.61)). As with *afd_shotrec*, two temporal sample rates are prescribed. The coarser one (`dt` on line 32) specifies the sample rate of the final seismogram, while the finer one (`dtstep` on line 33) is the time step of the modeling. The resulting seismogram, filtered with Ormsby parameters of [10 15 40 50], is shown in Figure 7.24b. Two small horizontal lines are superimposed to indicate the position of the channel. The lines are at the times of the top and bottom of the channel (0.312 and 0.355 s) and have the same width as the channel. The channel produces an extensive diffraction pattern showing scattered energy over the entire model.

Figure 7.25a shows a second exploding reflector seismogram of the channel model, which was calculated with the second-order approximate Laplacian (Eq. (4.60)). In comparison with the fourth-order seismogram of Figure 7.24b, this result is considerably less accurate, though it required only one half of the computation time. Even less accurate is the second-order solution shown in Figure 7.25b, which was obtained with a grid sample size of 10 m rather than the 5 m of the previous two figures. This result is almost unrecognizable as the response of the same structure. An obvious conclusion is that the results from finite-difference modeling should not be trusted without careful consideration of the algorithms and parameters.

As a slightly more complex example that will be useful in examining depth migration, consider the response of an anticline beneath a high-velocity wedge as shown in Figure 7.26a. Code Snippet 7.3.6 creates this velocity model and the fourth-order exploding reflector seismogram shown in Figure 7.27a. This result was created with a spatial grid size of 10 m. It might be tempting to trust this result as the proper physical response of the model, but that could lead to erroneous conclusions. For example, it might be concluded that the response of the anticline has a series of reverberations following the primary response. However, before leaping to conclusions, it is prudent to recreate the model with a finer spatial grid size. Figure 7.27b shows the result of rerunning Code Snippet 7.3.6 with the spatial grid size set to 5 m. The reverberations have vanished and the amplitude variation of the primary response of the anticline has changed considerably. Once again, it pays

Code Snippet 7.3.5 This code uses *afd_explode* to model the exploding reflector response of a small channel. Lines 1–8 build the velocity model, and lines 10–14 create a fourth-order exploding reflector seismogram. The velocity model construction uses *channelmodel*, which is one of several prebuilt models available in the *finitedif"* toolbox. The grid size is determined by dx, which is specified outside this snippet. The model is shown in Figure 7.24a. For the seismograms in Figures 7.24b and 7.25a, dx is 5 m, while for Figure 7.25b it is 10 m. The final input to *afd_explode* is a flag for the Laplacian approximation (see Eqs. (4.60) and (4.61)). It is also specified externally. Figure 7.24b uses the fourth-order approximation, while Figures 7.25a and 7.25b use a second-order one.

```
1   xmax=2500;zmax=1000; %maximum line length and maximum depth
2   vhigh=4000;vlow=2000; % high and low velocities
3   vrange=vhigh-vlow;
4   vs=vlow+(0:4)*vrange/5;
5   width=20;thk=50;zch=398;%channel dimensions
6   vch=vlow+vrange/6;%channel velocity
7   [vel,x,z]=channelmodel(dx,xmax,zmax,vhigh,vlow,zch,width,thk,...
8       vch,4,1,[100,200,271,398 zmax],vs);
9   %create the exploding reflector model
10  dt=.004; %temporal sample rate
11  dtstep=.001; %modeling step size
12  tmax=2*zmax/vlow; %maximum time
13  [seisfilt,seis,t]=afd_explode(dx,dtstep,-dt,tmax, ...
14                  vel,x,zeros(size(x)),[10 15 40 50],0,laplacian);
```

End Code

elmigcode/afdexample1.m

to be cautious when working with finite-difference synthetics. The only way to be certain is to fully understand the algorithm and how to change the parameters to increase the accuracy. When the accuracy parameters are increased by a factor of two and the response does not change significantly, then it is safe to conclude that the model is showing an accurate physical response. Of course, this can be a tedious process because increasing the accuracy will always increase the computation time.

7.3.3 Migration and Modeling with Normal Ray Tracing

Normal ray tracing, that is, the determination of the raypaths and traveltimes of normal-incidence rays, does not directly provide a complete seismic section. The determination of amplitudes would require solution of the geometric spreading equation, Eq. (6.73), in addition to the eikonal equation. However, even with this shortcoming, a great deal of useful information can be obtained by using normal ray tracing for migration or modeling.

The two key functions here are *normray* and *normraymig*, which do modeling and migration, respectively. Migration by normal-incidence ray tracing was described in Section 7.2.2, and normal ray modeling is essentially just the inverse process. The function *normraymig* accepts the three parameters defining a pick, $(x_0, t_0, \Delta t/\Delta x)$ (p. 363),

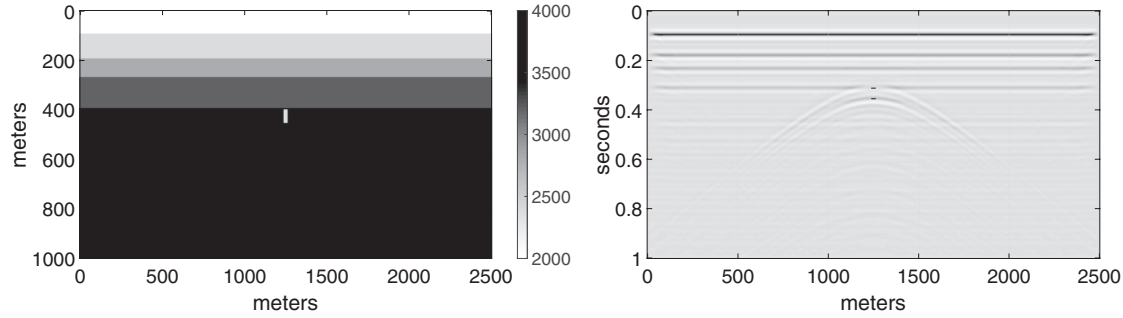


Figure 7.24a (left) The velocity model created by Code Snippet 7.3.5.

Figure 7.24b (right) The fourth-order (spatial) exploding reflector seismogram created on line 35 of Code Snippet 7.3.5. The top and bottom of the channel are marked by black lines.

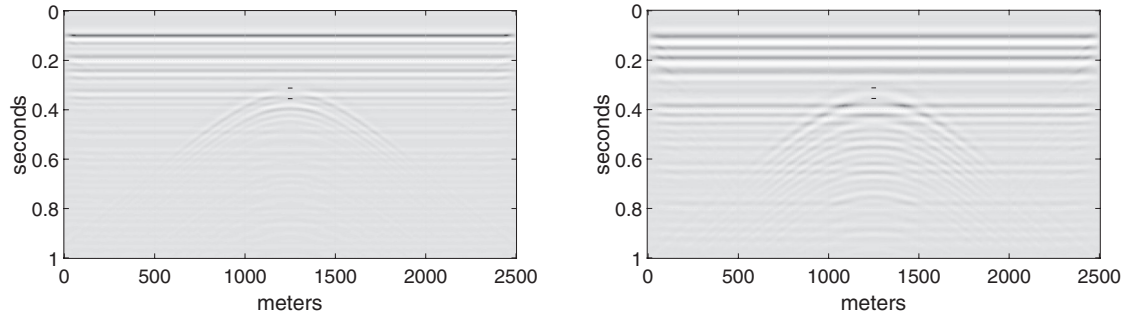


Figure 7.25a (left) Similar to the seismogram of Figure 7.24b except that the Laplacian was approximated with second-order finite differences.

Figure 7.25b (right) Similar to the seismogram of Figure 7.25a except that the spatial sample rate was 10 m rather than 5 m.

and then uses *shootrayvxz* (discussed in Section 6.12.2) to trace that ray down into a velocity model. The *plotimage* picking facility (Section 1.4.3) can be used to provide these picks for *normraymig* to migrate. The function *normraymig* returns the abstract ray vector, $\vec{r} = [\vec{x},\ \vec{p}]$ (see Eq. (6.90)), as an $n \times 4$ matrix. The first two columns are the (x, z) coordinates, while the second two are the horizontal and vertical slownesses. The number of rows is the number of time steps required to trace the ray. Thus the ray vector provides complete specification of the raypath. As a convenience, *normraymig* will plot the raypath in a user-specified figure.

The use of *normraymig*, as described in the previous paragraph, can become quite tedious if there are a great many picks to migrate. Therefore, *eventraymig* is provided to automatically migrate all of the picks stored in the global variable PICKS. The function *eventraymig* requires only a single input parameter, and that is the figure number to draw the raypaths in. However, prior to running either *normraymig* or *eventraymig*, the velocity model must be properly established using *rayvelmod* as described in Section 6.12.2.

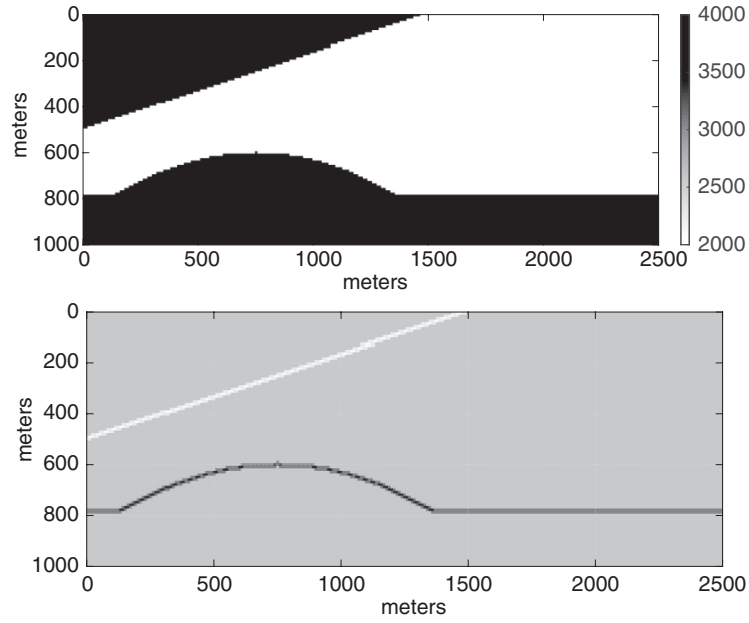


Figure 7.26a (top) An anticline beneath a high-velocity wedge. Black is 4000 m/s and white is 2000 m/s.

Figure 7.26b (bottom) The reflectivity corresponding to Figure 7.26a is shown. This was calculated with *afd_reflect*.

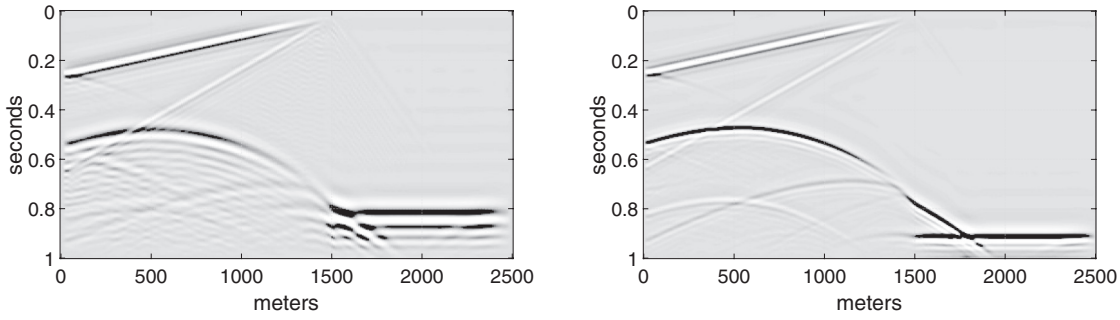


Figure 7.27a (left) The fourth-order exploding reflector seismogram corresponding to the model of Figure 7.26a. This model was created using a spatial grid size of 10 m.

Figure 7.27b (right) Similar to Figure 7.27a except that the spatial grid size was 5 m.

In Figure 7.28a, the exploding reflector seismogram of Figure 7.27b is shown with picks on the event from the anticline. These picks were made with the *plotimage* picking facility (Section 1.4.3) and so are automatically stored as a global variable for *eventraymig*. Prior to migrating these picks, and after running Code Snippet 7.3.6, the velocity matrices required for ray tracing were initialized with the command `rayvelmod(vel,dx)`, where `vel` is the velocity matrix and `dx` is the grid size. Then,

Code Snippet 7.3.6 This creates an exploding reflector model of an anticline beneath a high-velocity wedge. Lines 1–3 build the velocity model and lines 5–11 create a fourth-order exploding reflector seismogram. The results are shown in Figures 7.26a and 7.27a.

```
1   xmax=2500;zmax=1000; %maximum line length and maximum depth
2   vhigh=4000;vlow=2000; % high and low velocities
3   [vel,x,z]=wedgemodel(dx,xmax,zmax,vhigh,vlow);
4   %do a finite-difference model
5   dt=.004; %temporal sample rate
6   dtstep=.001;
7   tmax=2*zmax/vlow; %maximum time
8   %[w,tw]=wavemin(dt,30,.2); %minimum phase wavelet
9   %[w,tw]=ricker(dt,70,.2); %ricker wavelet
10  [seisfilt,seis,t]=afd_explode(dx,dtstep,dt,tmax, ...
11                  vel,x,zeros(size(x)),[5 10 40 50],0,laplacian);
```

End Code

elmigcode/afdexample2.m

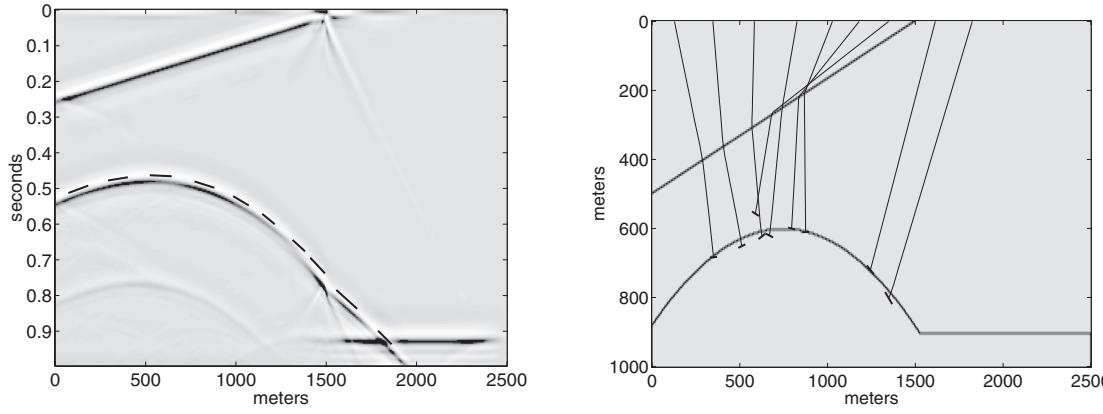


Figure 7.28a (left) The seismogram of Figure 7.27b is shown with picks on the image of the anticline.
Figure 7.28b (right) The picks of Figure 7.28a are shown migrated on top of the reflectivity section.

the command `eventraymig(figno)`, where `figno` is the MATLAB figure number of Figure 7.26a, caused the picks to be migrated, and the resulting raypaths are displayed in Figure 7.28b. At the termination of each raypath, a small line segment, perpendicular to the raypath, is drawn that indicates the implied reflector dip. (These do not appear perpendicular in Figure 7.28b, because the (x, z) axes do not have the same scale. The command `axis equal` will display any figure with equal scales on all axes.)

The relative accuracy of the migrations in Figure 7.28b is instructive. The picks have all migrated to positions near the anticline, but some have fallen short while others have gone too far. Among the obvious reasons for this are the difficulty in determining which phase (peak, trough, zero crossing, etc.) of the input waveform should be picked, and then

making a consistent pick at two points. A slight error in either case can result in a very large error in the final position. Since the material beneath the anticline has a high velocity, a pick that arrives at the anticline with a little time left (see Section 7.2.2) will continue a significant distance. Thus, small errors in the pick time can cause a large error in the result. Also, an error in picking $\Delta t/\Delta x$ will cause the initial trajectory of the ray to be incorrect. Since $\sin\theta_0 = 0.5 v_0\, \Delta t/\Delta x$, these emergence angle errors are also more significant in high-velocity material.

Migration by normal ray tracing can reveal a lot about the nature of a seismic dataset. Also instructive is the complementary process of normal-incidence raytrace modeling. This process is implemented in the function *normray*, which is the logical reverse of *normraymig*. The latter requires the pick specification of $(x_0, t_0, \Delta t/\Delta x)$, while the former needs the specification of the normal ray $(x_\mathrm{n}, z_\mathrm{n}, \theta_\mathrm{n})$. Here, $(x_\mathrm{n}, z_\mathrm{n})$ are the coordinates of the normal-incidence point and θ_n is the structural dip (in degrees) at the normal-incidence point. Though they are logically similar, it is convenient to use separate ray-tracing engines for these two tasks because they have different criteria for stopping the ray. In migration, the ray is terminated when it has used the available traveltime, while in modeling, it is stopped when it encounters the recording surface ($z = 0$). These ray-tracing engines are *shootrayvxz* and *shootraytosurf*, respectively.

As with the migration tools, it is tedious to invoke *normray* at the command line for each pick. Therefore, a convenience function, *eventraymod*, is provided that automatically models any picks found in the global variable PICKS. These picks are expected to have been made on a depth section, though no check is made to ensure this. Figure 7.29a shows the reflectivity section of Figures 7.26b and 7.28b with a series of picks, $(x_\mathrm{n}, z_\mathrm{n}, \theta_\mathrm{n})$, made on the anticline. Also shown are the normal-incidence raypaths (drawn by *normray*) to the surface. Figure 7.29b shows the modeled picks, $(x_0, t_0, \Delta t/\Delta x)$, on top of the seismic section of Figures 7.27b and 7.28a.

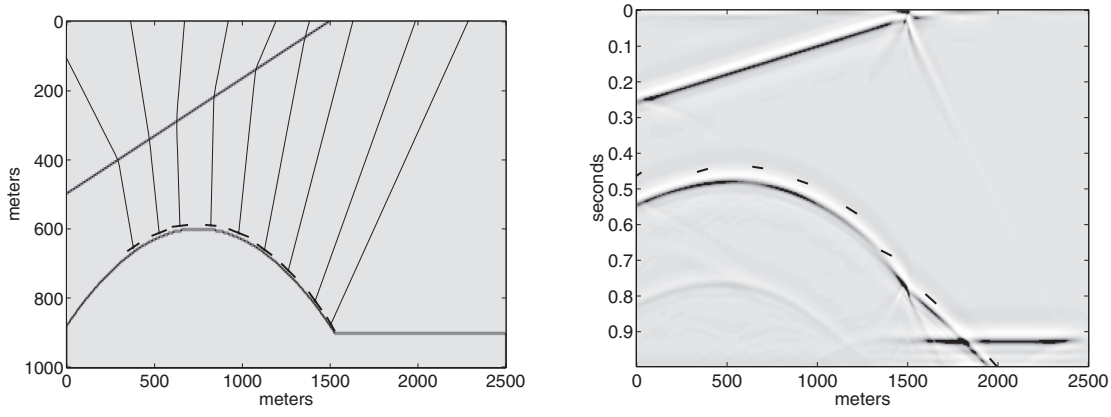


Figure 7.29a (left) The reflectivity section of Figure 7.26b is shown with picks on the anticline, and normal rays to the surface.
Figure 7.29b (right) The picks of Figure 7.29a are shown modeled on top of the seismic section of Figure 7.27b.

7.4 Fourier Methods

The Fourier methods are developed from an exact solution to the wave equation using Fourier transforms. They provide a high-fidelity migration that illustrates precisely how the migrated spectrum is formed. There are two fundamental approaches and many variations of each. The *f–k* migration (Stolt, 1978) is an exact solution to the migration problem for constant velocity. It is a direct method that is the fastest known migration technique. Phase-shift migration (Gazdag, 1978) is a recursive approach that uses a series of constant-velocity extrapolations to build a $v(z)$ migration.

7.4.1 *f–k* Migration

Stolt (1978) showed that the migration problem can be solved by Fourier transformation. Here, Stolt's solution will be developed from a formal solution of the wave equation using Fourier transforms. It will be developed in 2D and the direct 3D extension will be stated at the end.

Let $\psi(x, z, t)$ be a scalar wavefield that is a solution to

$$\nabla^2 \psi - \frac{1}{\hat{v}^2} \frac{\partial^2 \psi}{\partial t^2} = 0, \tag{7.40}$$

where $\hat{v}$ is the constant ERM velocity. It is desired to calculate $\psi(x, z, t = 0)$ given knowledge of $\psi(x, z = 0, t)$. The wavefield can be written as an inverse Fourier transform of its *f–k* spectrum as

$$\psi(x, z, t) = \int_{V_\infty} \phi(k_x, z, f) e^{2\pi i(k_x x - ft)} \, dk_x \, df, \tag{7.41}$$

where cyclical wavenumbers and frequencies are used and the Fourier transform convention uses a $+$ sign in the complex exponential for spatial components and a $-$ sign for temporal components. (The notation for the integration domain is explained in Section 7.1.3.) If Eq. (7.41) is substituted into Eq. (7.40), the various partial derivatives can be immediately brought inside the integral, where they can be readily evaluated. The result is

$$\int_{V_\infty} \left\{ \frac{\partial^2 \phi(z)}{\partial z^2} + 4\pi^2 \left[\frac{f^2}{\hat{v}^2} - k_x^2 \right] \phi(z) \right\} e^{2\pi i(k_x x - ft)} \, dk_x \, df = 0, \tag{7.42}$$

where the *f–k* dependence in $\phi(z)$ has been suppressed for simplicity of notation. The derivation of Eq. (7.42) does not require that $\hat{v}$ be constant; however, the next step does. If $\hat{v}$ is constant,[2] then the left-hand side of Eq. (7.42) is the inverse Fourier transform of the term in curly brackets. The uniqueness property of Fourier transforms (that there is a unique spectrum for a given function and vice versa) guarantees that if a function vanishes

[2] Actually, $\hat{v}(z)$ could be tolerated here. The necessary condition is that $\hat{v}$ must not depend upon x or t.

everywhere in one domain, it must do so in another. Put another way, the zero function has a zero spectrum. Thus, it results that

$$\frac{\partial^2 \phi(z)}{\partial z^2} + 4\pi^2 k_z^2 \phi(z) = 0, \tag{7.43}$$

where the wavenumber k_z is defined by

$$k_z^2 = \frac{f^2}{\hat{v}^2} - k_x^2. \tag{7.44}$$

Equation (7.44) is called the *dispersion relation for scalar waves*, though the phrase is somewhat misleading since there is no dispersion in this case.

Equations (7.43) and (7.44) are a complete reformulation of the problem in the *f–k* domain. The boundary condition is now $\phi(k_x, z = 0, f)$, which is the Fourier transform, over (x, t), of $\psi(x, z = 0, t)$. Equation (7.43) is a second-order ordinary differential equation for fixed f and k. Either of the functions $e^{\pm 2\pi i k_z z}$ solves it exactly, as is easily verified by substitution. Thus the unique, general solution can be written as

$$\phi(k_x, z, f) = A(k_x, f) e^{2\pi i k_z z} + B(k_x, f) e^{-2\pi i k_z z}, \tag{7.45}$$

where $A(k_x, f)$ and $B(k_x, f)$ are arbitrary functions of f and k to be determined from the boundary condition(s). The two terms on the right-hand side of Eq. (7.45) have the interpretation of a downgoing wavefield, $A(k_x, f) e^{2\pi i k_z z}$, and an upgoing wavefield, $B(k_x, f) e^{-2\pi i k_z z}$. This can be seen by substituting Eq. (7.45) into Eq. (7.41) and determining the direction of motion of the individual Fourier plane waves as described in Section 6.9. It should be recalled that z increases downward.

Given only one boundary condition, $\phi(k_x, z = 0, f)$, it is now apparent that this problem cannot be solved unambiguously. It is a fundamental result from the theory of partial differential equations that *Cauchy* boundary conditions (e.g., knowledge of both ψ and $\partial_z \psi$) are required on an open surface in order for the wave equation to have a unique solution. Since this is not the case here, the migration problem is said to be ill-posed. If both conditions were available, A and B could be found as the solutions to

$$\phi(z = 0) \equiv \phi_0 = A + B \tag{7.46}$$

and

$$\frac{\partial \phi}{\partial z}(z = 0) \equiv \phi_{z0} = 2\pi i k_z A - 2\pi i k_z B. \tag{7.47}$$

When faced with the need to proceed to a solution despite the fact that the stated problem does not have a unique solution, a common approach is to assume some limited model that removes the ambiguity. The customary assumption of *one-way waves* achieves this end. That is, $\psi(x, z, t)$ is considered to contain only upgoing waves. This allows the solution

$$A(k_x, f) = 0 \quad \text{and} \quad B(k_x, f) = \phi_0(k_x, f) \equiv \phi(k_x, z = 0, f). \tag{7.48}$$

Then, the ERM wavefield can be expressed as the inverse Fourier transform

$$\psi(x,z,t) = \int_{V_\infty} \phi_0(k_x,f) e^{2\pi i(k_x x - k_z z - ft)}\, dk_x\, df. \tag{7.49}$$

The migrated solution is

$$\psi(x,z,t=0) = \int_{V_\infty} \phi_0(k_x,f) e^{2\pi i(k_x x - k_z z)}\, dk_x\, df. \tag{7.50}$$

Equation (7.50) gives a migrated depth section as a double integration of $\phi_0(k_x,f)$ over f and k_x. Though formally complete, it has the disadvantage that only one of the integrations, that over k_x, is a Fourier transform that can be done rapidly as a numerical FFT. The f integration is not a Fourier transform, because the Fourier kernel $e^{-2\pi ft}$ was lost when the imaging condition (setting $t = 0$) was invoked. Inspection of Eq. (7.50) shows that another complex exponential, $e^{-2\pi ik_z z}$, is available. Stolt (1978) suggested a change of variables from (k_x,f) to (k_x,k_z) to obtain a result in which both integrations are Fourier transforms. The change of variables is actually defined by Eq. (7.44), which can be solved for f to give

$$f = \hat{v}\sqrt{k_x^2 + k_z^2}. \tag{7.51}$$

Performing the change of variables from f to k_z according to the rules of calculus transforms Eq. (7.50) into

$$\psi(x,z,t=0) = \int_{V_\infty} \phi_m(k_x,k_z) e^{2\pi i(k_x x - k_z z)}\, dk_x\, dk_z, \tag{7.52}$$

where

$$\phi_m(k_x,k_z) \equiv \frac{\partial f(k_z)}{\partial k_z}\phi_0(k_x,f(k_z)) = \frac{\hat{v}k_z}{\sqrt{k_x^2+k_z^2}}\phi_0(k_x,f(k_z)). \tag{7.53}$$

Equation (7.52) is Stolt's expression for the migrated section and forms the basis for the *f–k* migration algorithm. The change of variables has recast the algorithm into one that can be accomplished with FFTs doing all of the integrations. Equation (7.53) results from the change of variables and is a prescription for the construction of the (k_x,k_z) spectrum of the migrated section from the *f–k* spectrum of the ERM seismogram.

Many of the important properties of poststack migration can be discerned from Stolt's result. First, notice that k_z defined through

$$k_z = \sqrt{\frac{f^2}{\hat{v}^2} - k_x^2} \tag{7.54}$$

is real-valued when $|f/k_x| \geq \hat{v}$ and is otherwise imaginary. Only for real k_z will Eq. (7.45) correspond to traveling waves in the positive and negative z directions. For complex k_z, $e^{\pm 2\pi ik_z z}$ becomes a real exponential that either grows or decays; however, on physical grounds, growing exponentials must be excluded. Given a value for $\hat{v}$, this dual behavior

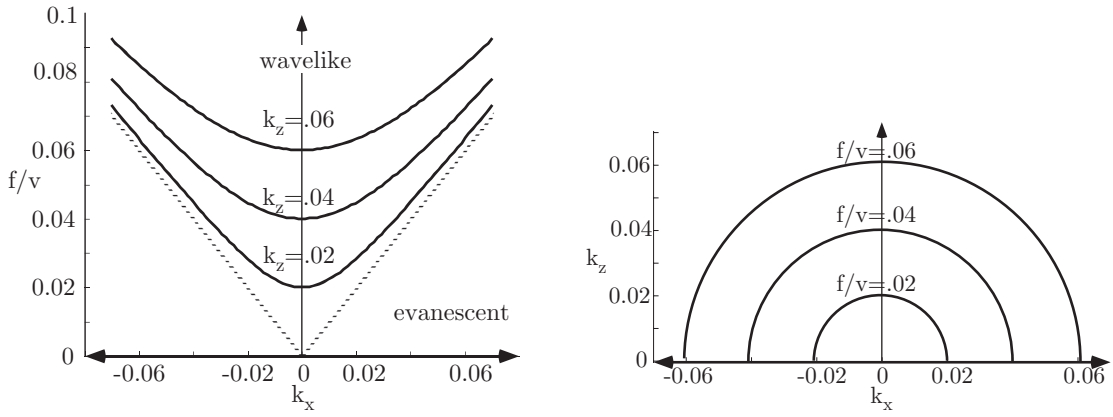


Figure 7.30a (left) The space $(f/v, k_x)$ is shown contoured with values of k_z from Eq. (7.54). The dashed lines are the boundary between the wavelike and evanescent regions.

Figure 7.30b (right) The space (k_x, k_z) is shown with contours of f/v as given by Eq. (7.51).

of k_z divides *f–k* space into two regions, as was discussed from another perspective in Section 6.11.1. The region where $|f/k_x| \geq \hat{v}$ is called the *wavelike* or *body wave* region, and where $|f/k_x| < \hat{v}$ it is called the *evanescent* region. Equation (7.54) is sometimes called a dispersion relation for one-way waves, since the choice of the plus sign in front of the square root generates upward-traveling waves in $e^{\pm 2\pi i k_z}$, while the minus sign generates downward-traveling waves.

The geometric relationships between the spaces of $(k_x, f/v)$ and (k_x, k_z) are shown from two different perspectives in Figures 7.30a and 7.30b. In $(k_x, f/v)$ space, the lines of constant k_z are hyperbolas that are asymptotic to the dashed boundary between the wavelike and evanescent regions. In (k_x, k_z) space, the curves of constant f/v are semicircles. At $k_x = 0$, $k_z = f/v$, so these hyperbolas and semicircles intersect when the plots are superimposed.

The spectral mapping required in Eq. (7.53) is shown in Figure 7.31. The mapping takes a constant-f slice of (k_x, f) space to a semicircle in (k_x, k_z) space. Each point on the f slice maps at constant k_x, which is directly down in the figure. It is completely equivalent to view the mapping as a flattening of the k_z hyperbolas of Figure 7.30a. In this sense, the mapping is conceptually similar to the NMO removal in the time domain, though here the samples being mapped are complex-valued. That the spectral mapping happens at constant k_x is a mathematical consequence of the fact that k_x is held constant while the f integral in Eq. (7.50) is evaluated. Conceptually, it can also be viewed as a consequence of the fact that the ERM seismogram and the migrated section must agree at $z = 0$ and $t = 0$.

On a numerical dataset, this spectral mapping is the major complexity of the Stolt algorithm. Generally, it requires an interpolation in the *f–k* domain, since the spectral values that map to grid nodes in (k_x, k_z) space cannot be expected to come from grid nodes in (k_x, f) space. In order to achieve the significant computation speed that is considered the strength of the Stolt algorithm, it turns out that the interpolation must always be approximate. This causes *artifacts* in the final result. This issue will be discussed in more detail in Section 7.4.2.

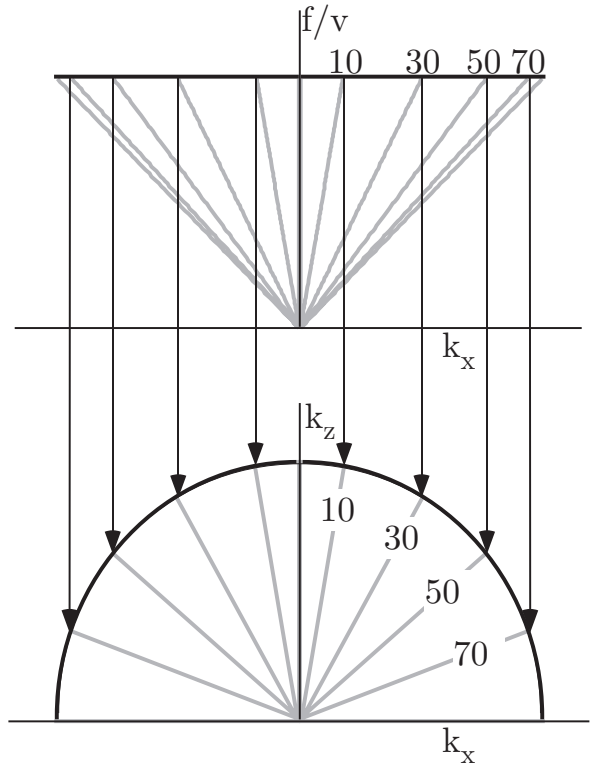


Figure 7.31 The mapping from (k_x, f) (top) to (k_x, k_z) (bottom). A line of constant f is mapped, at constant k_x, to a semicircle in (k_x, k_z). The compressed apparent-dip spectrum in $(f/v, k_x)$ space unfolds into the uniform fan in (k_x, k_z) space. The numbers on each space indicate dips in degrees.

The creation of the migrated spectrum also requires that the spectrum be scaled by $\hat{v}k_z/\sqrt{k_x^2 + k_z^2}$ as it is mapped (Eq. (7.53)). In the constant-velocity medium of this theory, $\sin\delta = \hat{v}k_x/f$ (where δ is the scattering angle), from which it follows that $\cos\delta = \hat{v}k_z/f = k_z/\sqrt{k_x^2 + k_z^2}$. Thus the scaling factor is proportional to $\cos\delta$ and therefore ranges from unity to zero as δ goes from 0°to 90°. This scaling factor compensates for the "spectral compression" that is a theoretical expectation in the ERM seismogram. Recall the migrator's equation (Eq. (7.35)), which relates apparent angles in $(k_x, f/v)$ space to real angles in (k_x, k_z) space. If there is uniform power at all angles in (k_x, k_z) space, then the migrator's equation predicts a spectral compression in $(k_x, f/v)$ (with consequent increasing power) near 45° of apparent dip. As shown in Figure 7.31, it is as though (k_x, k_z) space is an oriental fan that has been folded to create $(k_x, f/v)$ space. Migration must then unfold this fan. *f–k* migration is called a *steep-dip* method because it works correctly for all scattering angles from 0° to 90°.

The *f–k* migration algorithm just described is limited to constant velocity, though it is exact in this case. Its use of Fourier transforms for all of the numerical integrations means

that it is computationally very efficient. Though it has been used for many years in practical applications, its restriction to constant velocity is often unacceptable, so that it is gradually being replaced by more flexible, though usually slower, methods. Today, one major virtue still remains and that is the conceptual understanding it provides. The description of the construction of the migrated spectrum will provide the basis for a realistic theory of seismic resolution in Section 7.7.

7.4.2 A MATLAB Implementation of *f–k* Migration

The *f–k* migration algorithm just described is a three-step process:

1. *Forward f–k transform.* This is done on the unmigrated data after all preconditioning such as spectral whitening, band-pass filtering, and noise reduction.
2. *Spectral mapping.* The spectral mapping and scaling described by Eq. (7.53) requires a careful interpolation process.
3. *Inverse f–k transform.* This must be the exact inverse of the transform done in the first step.

This process is formalized in the MATLAB function `fkmig`. This function has the external interface `[seismig,tmig,xmig] = fkmig(seis,t,x,v,params)`. Here the first four input variables are simply the seismic matrix, its time coordinate vector, its x coordinate vector, and a scalar giving the velocity of migration. The final input variable is a vector of parameters that control various aspects of the migration. This vector, `params`, has length 13 and affords control over the spatial and temporal zero pads, the maximum dip to migrate, the maximum frequency to migrate, the type of spectral interpolation, and the kind of progress messages that are written to the screen. Consult the online help for `fkmig` for a complete description. Usually, the default actions for all of these behaviors are acceptable and `params` can be omitted. Occasionally, it will be desired to program one or more elements of `params`. This can be done while still defaulting the others by first creating a vector of thirteen NaNs (e.g., `params=nan*1:13;`) and then setting a few elements of `params` to specific values.

As a first example, consider the migration of the synthetic sections shown in Figures 7.19a and 7.19b. This is very simply accomplished with the code in Code Snippet 7.4.1, with the results shown in Figures 7.32a and 7.32b. These figures are displayed with slight clipping to make their more subtle details visible. In Code Snippet 7.4.1, the variable `seis` refers to the data of Figure 7.19a, while `seis2` refers to Figure 7.19b. Figure 7.19a contains the image of a dipping reflecting segment without diffractions, while Figure 7.19b contains a similar image except that diffractions are present. The resulting migrations make clear the advantage (indeed, the necessity) of modeling with diffractions.

Code Snippet 7.4.1 This code uses *fkmig* to migrate the sections shown in Figures 7.19a and 7.19b. The results are shown in Figures 7.32a and 7.32b.

```
seismig=fkmig(seis,t,x,v);
plotimage(seismig,t*v/2,x);
xlabel('meters');ylabel('meters');
seismig2=fkmig(seis2,t,x,v);
plotimage(seismig2,t*v/2,x);
xlabel('meters');ylabel('meters');
```

End Code

elmigcode/fkmig_ex1.m

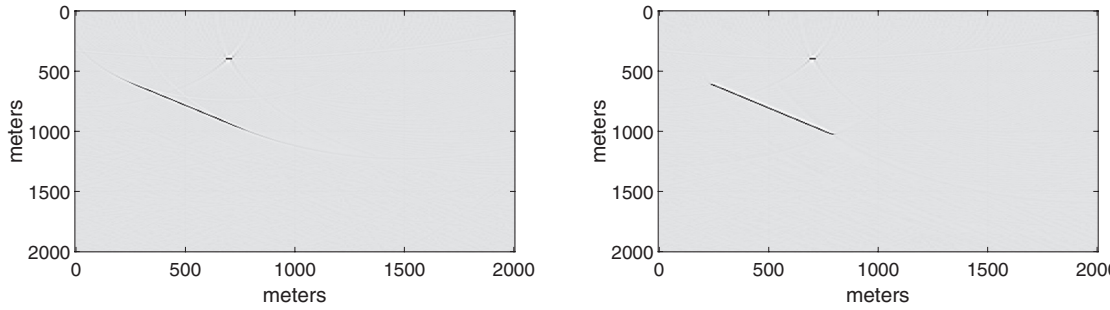


Figure 7.32a (left) The result of the *f*–*k* migration of the data of Figure 7.19a.
Figure 7.32b (right) The result of the *f*–*k* migration of the data of Figure 7.19b.

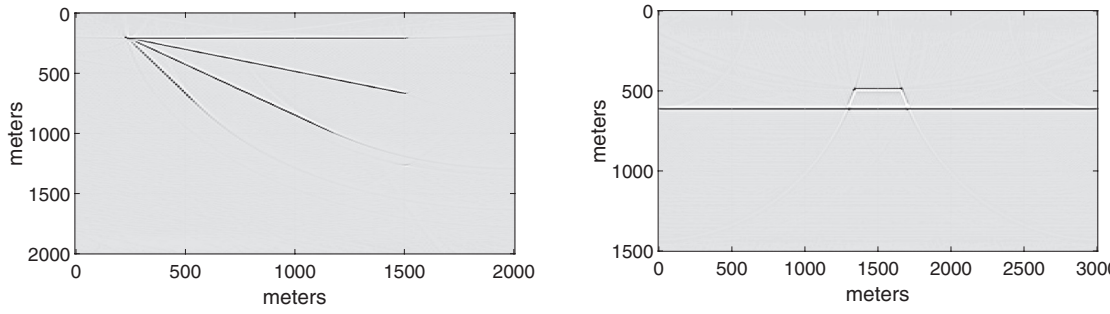


Figure 7.33a (left) The result of the *f*–*k* migration of the data of Figure 7.20a.
Figure 7.33b (right) The result of the *f*–*k* migration of the data of Figure 7.21a.

Only when diffractions are included are the edges of the reflecting segment imaged with crisp definition.

As further examples, the *f*–*k* migrations of the data of Figures 7.20a, 7.21a, 7.22a, and 7.23b can be migrated in similar fashion. The results are shown in Figures 7.33a, 7.33b, 7.34a, and 7.34b.

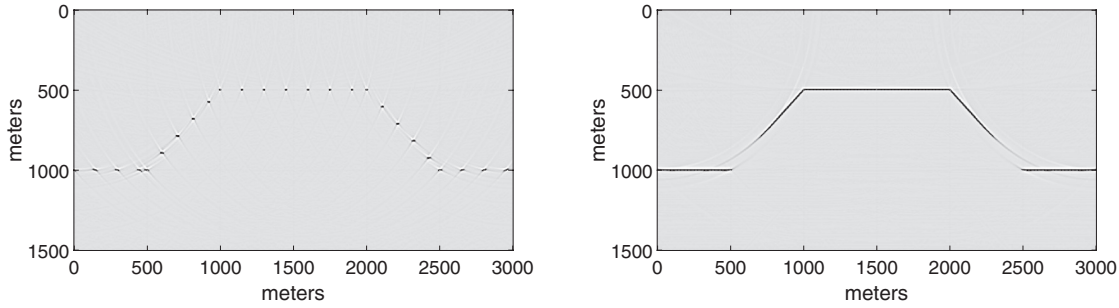


Figure 7.34a (left) The result of the *f–k* migration of the data of Figure 7.22a.
Figure 7.34b (right) The result of the *f–k* migration of the data of Figure 7.23b.

It is apparent that *fkmig* creates high-quality migrations of constant-velocity synthetics, but it is instructive to examine the code to see precisely how this is done. Code Snippet 7.4.2 contains an excerpt from the source code of *fkmig* that illustrates the steps of basic geophysical importance. The forward *f–k* transform is accomplished on line 2 by calling *fktran*. Prior to this, the seismic matrix, `seis`, has had zero pads attached in both x (columns) and t (rows). The variables `tnew` and `xnew` are time coordinate and x coordinate vectors that describe the padded seismic matrix. Similarly, `nsampnew` and `ntrnew` are the number of samples per trace and the number of traces after padding. The function *fktran* is a geophysical wrapper around MATLAB's built-in two-dimensional FFT `fft2`. This means that, in addition to calling `fft2` to compute the *f–k* spectrum, `fkspec`, of `seis`, it does some useful things such as calculating the frequency and wavenumber coordinate vectors `f` and `kx`. Also, because the input matrix is real-valued, it returns only the nonnegative temporal frequencies. This is possible because the negative frequencies can be deduced from the positive ones by a symmetry argument. Technically, if $\phi(k_x,f)$ is the *f–k* transform of the real-valued wavefield $\psi(x,t)$, then it can be shown that $\phi(k_x,f)$ has the symmetry $\phi(k_x,f) = \bar{\phi}(-k_x,-f)$, where the overbar indicates the complex conjugate. Both positive and negative wavenumbers are required since, after the temporal Fourier transform, the matrix is no longer real-valued. Working only with the nonnegative temporal frequencies is more efficient because less memory and fewer calculations are required. Also, it requires more care to formulate an algorithm correctly to process both positive and negative frequencies. If the processed spectrum does not have the correct conjugate symmetry, then its inverse transform will result in a complex-valued seismic matrix. It is easier to process only the nonnegative frequencies and calculate the negative ones as needed from the symmetry condition. The function *fktran* has a companion inverse *ifktran* (invoked on line 34) that creates the negative temporal frequencies from the nonnegative ones and then calls `fft2`.

Continuing with Code Snippet 7.4.2, line 3 calculates the exploding reflector velocity that is required in the theory. The major computation loop is over k_x (the columns of `fkspec`) and extends from line 9 to line 32. Each iteration through the loop converts one column of `fkspec`, representing ϕ_0 of Eq. (7.53), into a column vector representing ϕ_m.

Code Snippet 7.4.2 This is an excerpt from the code of *fkmig* that shows the steps of major importance. The actual *fkmig* source code contains 319 lines that handle the many nontechnical details required for a useful code.

```
1   %forward fk transform
2   [fkspec,f,kx] = fktran(seis,tnew,xnew,nsampnew,ntrnew,0,0);
3   ve = v/2; %exploding reflector velocity
4   %compute kz
5   dkz= df/ve;
6   kz = ((0:length(f)-1)*dkz)';
7   kz2=kz.^2;
8   %now loop over wavenumbers
9   for j=1:length(kx)
10  % apply masks
11      tmp=fkspec(:,j).*fmask.*dipmask;
12  %compute f's which map to kz
13      fmap = ve*sqrt(kx(j)^2 + kz2);
14      ind=find(fmap<=fmaxmig);
15  %now map samples by interpolation
16      fkspec(:,j) = zeros(length(f),1); %initialize output spectrum
17      if( ~isempty(ind) )
18          %compute cosine scale factor
19          if( fmap(ind(1))==0)
20              scl=ones(size(ind));
21              li=length(ind);
22              scl(2:li)=ve*kz(ind(2:li))./fmap(ind(2:li));
23          else
24              scl=ve*kz(ind)./fmap(ind);
25          end
26          %complex sinc interpolation
27          fkspec(ind,j) = scl.*csinci(tmp,f,fmap(ind),...
28              [lsinc,ntable]);
29      end
30      if( floor(j/kpflag)*kpflag == j)
31          disp(['finished wavenumber ' int2str(j)]);
32      end
33  end
34  %inverse transform
35  [seismig,tmig,xmig]=ifktran(fkspec,f,kx);
```

End Code

elmigcode/fkmig_excerpt.m

Prior to the loop, lines 5 and 6 compute the vertical coordinate vector, `kz`, for the matrix representing ϕ_m (`df` is the frequency sample rate). On line 7, k_z^2 is precomputed, as this is needed in every loop. The first action in the loop (line 11) is to apply frequency and dip *masks* to the relevant column of `fkspec`, and the result is stored in the temporary vector `tmp`. The calculation of these masks is not shown in this example. They are simply vectors of the same size as a column of `fkspec` whose entries range between zero and one. The frequency mask, `fmask`, is precomputed outside the loop but the dip mask, `dipmask`, must

be recalculated with every iteration. These masks are designed to attenuate the frequencies and dips that are not of interest. For example, it is customary to migrate only frequencies below a certain maximum, $f_{\max}$. Thus a simple `fmask` could be unity for $f \le f_{\max}$ and zero for $f > f_{\max}$. However, from a signal-processing perspective, such an abrupt cutoff is not desirable, so a better `fmask` might be

$$f_{\text{mask}} = \begin{cases} 1, f \le f_{\max} - f_{\text{wid}}, \\ 3pt\dfrac{1}{2} + \dfrac{1}{2}\cos\left(\pi\dfrac{f - f_{\max} + f_{\text{wid}}}{f_{\text{wid}}}\right), f_{\max} - f_{\text{wid}} < f \le f_{\max}, \\ 0, f > f_{\max}. \end{cases} \tag{7.55}$$

This defines a *raised-cosine ramp* of width f_{wid} from unity at $f = f_{\max} - f_{\text{wid}}$ to zero at $f = f_{\max}$. The dip mask must be recomputed at every loop iteration because the frequency corresponding to a given dip changes as k_x changes according to $f = k_x \hat{v} / \sin\theta$. Like the frequency mask, the dip mask should use a raised-cosine ramp from zero at $\theta_{\max}$ to unity at $\theta_{\max} - \theta_{\text{wid}}$. Thus `dipmask` attenuates low frequencies, while `fmask` attenuates high frequencies.

Next, on line 13 in Code Snippet 7.4.2, the frequencies that map to each output k_z called `fmap`, are calculated via Eq. (7.51). Since the spectral mapping generally lowers frequencies (see Figure 7.31), many of these frequencies will, in general, be greater than the maximum desired frequency to migrate. Thus, on line 14, MATLAB's powerful `find` command is used to identify the entries in the vector `fmap` that are lower than the maximum frequency to migrate, `fmapmig`. (`fmapmig` corresponds to $f_{\max}$ in Eq. (7.55).) At this stage, the vector `ind` is a vector of indices into `fmap` that points to those frequencies that will be migrated.

On line 16, the current column of `fkspec` is set to zero in preparation for the actual migration that happens in the if-block from line 17 to line 28. Lines 18–25 compute the cosine scale factor that occurs in Eq. (7.53), with a special case for $f = 0$ to avoid division by zero. Line 27 does the actual spectral mapping and applies the cosine scale factor. The spectral mapping is accomplished by a *sinc function interpolation* that is optimized for complex-valued spectra, by the function `csinci`.

On lines 29–31, a progress message is printed. Though not strictly necessary, this is considered "good form" because there are few things more disconcerting than waiting for a computer to finish a long calculation without any indication that progress is occurring. A message is printed only after every `kpflag` wavenumbers have been processed. This is controllable through the input `params` vector.

Finally, after the completion of the loop on line 34, the migrated spectrum is inverse transformed. Generally, the zero pads are then removed, though this is not shown.

As a last example, consider the computation of the discrete *f–k* spectra of one of the preceding examples before and after migration. This should produce a graphical confirmation of the mapping of Figure 7.31. This is very simply done using `fktran`. Specifically, the case of the synthetic of Figure 7.22a and its migration in Figure 7.34a is shown. If `seis` is the unmigrated seismic matrix, then the command `[fkspec,f,kx]=fktran(seis,t,x)` computes the complex-valued *f–k* spectrum and `plotimage(abs(fkspec),f,kx)` produces the result shown in Figure 7.35a. In a similar manner, the (k_x, k_z) spectrum after

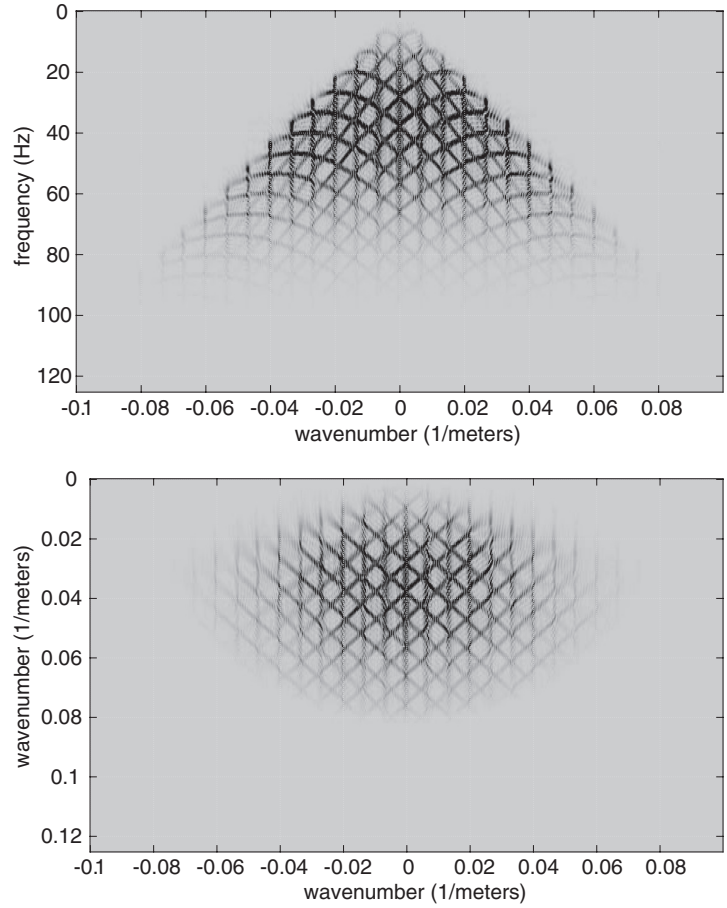


Figure 7.35a (top) The *f–k* spectrum of the data of Figure 7.22a.
Figure 7.35b (bottom) The (k_x, k_z) spectrum of the data of Figure 7.34a.

migration can be computed and is shown in Figure 7.35b. Comparison of these figures shows that the spectral mapping has been performed as described.

7.4.3 *f–k* Wavefield Extrapolation

Stolt (1978) provided an approximate technique to adapt *f–k* migration to $v(z)$. This method used a premigration step called the *Stolt stretch*, which was followed by an *f–k* migration. The idea was to perform a one-dimensional time-to-depth conversion with $v(z)$ and then convert back to a pseudo-time with a constant *reference velocity*. The *f–k* migration was then performed with this reference velocity. (Stolt actually recommended the time-to-depth conversion be done with a special velocity function derived from $v(z)$, called the Stolt velocity.) This method is now known to progressively lose accuracy with increasing dip and has lost favor.

A technique that can handle all scattering angles in $v(z)$ is the phase-shift method of Gazdag (1978). Unlike the direct *f–k* migration, phase shift is a recursive algorithm that treats $v(z)$ as a system of constant-velocity layers. In the limit as the layer thickness shrinks to infinitesimal, any $v(z)$ variation can be modeled. The method can be derived starting from Eq. (7.49). Considering the first velocity layer only, this result is valid for any depth within that layer provided that $\hat{v}$ is replaced by the first layer velocity, $\hat{v}_1$. If the thickness of the first layer is Δz_1, then the ERM wavefield just above the interface between layers 1 and 2 can be written as

$$\psi(x, z = \Delta z_1, t) = \int_{V_\infty} \phi_0(k_x, f) e^{2\pi i(k_x x - k_{z1}\,\Delta z_1 - ft)}\, dk_x\, df, \tag{7.56}$$

where

$$k_{z1} = \sqrt{\frac{f^2}{\hat{v}_1^2} - k_x^2}. \tag{7.57}$$

Equation (7.56) is an expression for *downward continuation* or *extrapolation* of the ERM wavefield to the depth Δz_1. The extrapolated wavefield is distinguished from the surface recorded wavefield by the presence of the term $e^{2\pi i k_{z1}\,\Delta z_1}$ under the integral sign, which is a specific form of the Fourier *extrapolation operator*, $e^{2\pi i k_z\,\Delta z}$. Any extrapolated wavefield is a temporal seismogram more akin to a surface recording than to a migrated depth section. In the phase-shift method, as with any recursive technique, the migrated depth section is built little by little from each extrapolated seismogram. The extrapolated wavefield is a simulation of what would have been recorded had the receivers actually been at $z = \Delta z$ rather than $z = 0$. Since any extrapolated section intersects the depth section at $(z = \Delta z, t = 0)$, each extrapolation can contribute one depth sample to the migration (see Figure 7.18). This process of evaluating the extrapolated section at $t = 0$ was discussed in Section 7.2.6 as the poststack imaging condition.

For the wavelike portion of *f–k* space, the extrapolation operator has unit amplitude and a phase of $2\pi k_z\,\Delta z$. For evanescent spectral components, it is a real exponential $e^{\pm 2\pi |k_z|\,\Delta z}$. Forward wavefield propagation must obey physical law and propagate evanescent spectral components using the minus sign in the exponent, e.g., $e^{-2\pi |k_z|\,\Delta z}$. Therefore, inverse wavefield extrapolation, as is done for migration, should use $e^{+2\pi |k_z|\,\Delta z}$. However, this inversion of evanescent spectral components is a practical impossibility because they have decayed far below the noise level in forward propagation. The practical approach is to use $e^{-2\pi |k_z|\,\Delta z}$ for the evanescent spectral components for both forward and inverse extrapolation. Even more practical is to simply zero the evanescent spectral components on inverse extrapolation. In all that follows, it is assumed that $e^{2\pi i k_z\,\Delta z}$has one of these two practical extensions implemented for evanescent spectral components.

The wavefield extrapolation expression (Eq. (7.56)) is more simply written in the Fourier domain to suppress the integration that performs the inverse Fourier transform:

$$\phi(k_x, z = \Delta z_1, f) = \phi_0(k_x, f) e^{2\pi i k_{z1}\,\Delta z_1}. \tag{7.58}$$

Now consider a further extrapolation to estimate the wavefield at the bottom of layer 2 ($z = \Delta z_1 + \Delta z_2$). This can be written as

$$\phi(k_x, z = \Delta z_1 + \Delta z_2, f) = \phi(k_x, z = \Delta z_1, f) T(k_x, f) e^{2\pi i k_{z2} \Delta z_2}, \tag{7.59}$$

where $T(k_x, f)$ is a correction factor for the transmission loss suffered by the upgoing wave as it crossed from layer 2 into layer 1. If transmission loss correction is to be incorporated, then the data must not have had such amplitude corrections applied already. This is extremely unlikely because seismic data is generally treated with a statistical amplitude balance that will compensate for transmission losses. Also, correcting for transmission losses at each extrapolation step can be numerically unstable because the correction factors are generally greater than unity. Consequently, it is customary to set $T(k_x, f) = 1$. With this, and incorporating Eq. (7.58), Eq. (7.59) becomes

$$\phi(k_x, z = \Delta z_1 + \Delta z_2, f) = \phi_0(k_x, f) e^{2\pi i \sum_{m=1}^{2} k_{zm} \Delta z_m}, \tag{7.60}$$

which has the immediate generalization to n layers

$$\phi\left(k_x, z = \sum_{m=1}^{n} \Delta z_m, f\right) = \phi_0(k_x, f) e^{2\pi i \sum_{m=1}^{n} k_{zm} \Delta z_m}. \tag{7.61}$$

In the limit as $\Delta z_m \to dz$, the summation in the extrapolator phase becomes an integral with the result

$$\phi(k_x, z, f) = \phi_0(k_x, f) e^{2\pi i \int_0^z k_z(z')\, dz'}, \tag{7.62}$$

where $k_z(z) = \sqrt{f^2/\hat{v}(z)^2 - k_x^2}$. This result is known as a first-order WKBJ[3] solution and can be derived as an approximate solution to the scalar wave equation for $\hat{v}(z)$ (Aki and Richards, 1980). The second-order WKBJ solution arises when transmission losses are considered.

Equation (7.61) expresses what can be realized in a practical implementation of recursive wavefield extrapolation, while Eq. (7.62) is the theoretical limit of the process. The theoretically correct phase shift from z_1 to z_2 is 2π times the area under the $k_z(z)$ curve between these depths. It turns out that this result is true in one, two, or three dimensions but the form of $k_z(z)$ changes. In one dimension, $k_z(z) = f/\hat{v}(z)$ and $\int_0^z k_z(z')\, dz' = f \int_0^z \hat{v}(z')^{-1}\, dz' = f\tau(z)$, where $\tau(z)$ is the vertical traveltime. In three dimensions, $k_z(z) = \sqrt{f^2/\hat{v}(z)^2 - k_x^2 - k_y^2}$ and the integral of $k_z(z)$ has no immediate simplification.

An interpretation of the extrapolation phase shift is suggested by incorporating $\sin\theta = \hat{v}k_x/f$, with the result

$$2\pi k_z\, \Delta z = 2\pi\, \Delta z \sqrt{\frac{f^2}{\hat{v}^2} - k_x^2} = 2\pi f \frac{\Delta z}{\hat{v}} \sqrt{1 - \left[\frac{\hat{v}k_x}{f}\right]^2} = 2\pi f \tau \cos\theta, \tag{7.63}$$

[3] The name WKBJ is derived from the initials of Wentzel, Kramers, Brillouin, and Jeffreys, who all derived it independently. The first three were quantum theorists, while Jeffreys was a geophysicist.

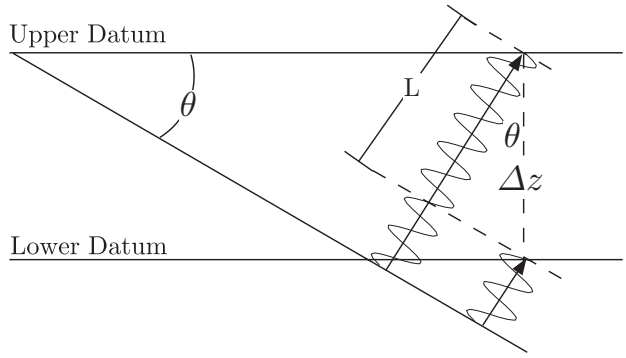


Figure 7.36 A geometric interpretation of the extrapolation phase shift. A dipping reflector emits a monochromatic signal that is received at two different datums. The extrapolation phase shift is the phase difference between this signal as it arrives at a specific (x_0, z_0) location on the upper datum and the signal as it arrives at the lower datum at the same horizontal coordinate, i.e., at $(x_0, z_0 + \Delta z)$.

where $\tau = \Delta z/\hat{v}$ has been used. Thus, for constant velocity, the extrapolation phase shift is scattering-angle dependent and ranges from a maximum of $2\pi f\tau$ at 0° to a minimum of 0 at 90°. Figure 7.36 shows a geometric interpretation of this phase shift. For a monochromatic plane wave, the phase shift extrapolator corrects for the phase difference between the wave's arrival at the upper datum at (x_0, z_0) and its arrival at the lower datum at $(x_0, z_0 + \Delta z)$. This phase difference is just 2π times the number of wavelengths that fit into the path difference L, that is,

$$\text{phase difference} = 2\pi \frac{L}{\lambda} = 2\pi \frac{\Delta z \cos\theta}{\hat{v}/f} = 2\pi \frac{f\Delta z}{\hat{v}} \cos\theta, \tag{7.64}$$

which is the same result as Eq. (7.63). For a recursive sequence of many constant-velocity phase shifts, the geometric interpretation is as shown in Figure 7.37.

The extrapolation operator is often separated into two terms, one that accomplishes a bulk time shift and another that accomplishes dip-dependent focusing. The bulk time delay operator is simply the extrapolation operator evaluated at $k_x = 0$ (i.e., zero dip) and is called the *static shift operator* or *thin lens operator*. The phase shift it applies is simply $\mu_s = 2\pi f \Delta z/\hat{v}$. Denoting the total phase shift by μ, this suggests that the focusing phase shift is simply $\mu_f = \mu - \mu_s$. Thus,

$$\mu_f = \mu - \mu_s = 2\pi\ \Delta z \sqrt{\frac{f^2}{\hat{v}^2} - k_x^2} - \frac{2\pi f \Delta z}{\hat{v}} = \frac{2\pi f \Delta z}{\hat{v}} \left[\sqrt{1 - \frac{k_x^2 \hat{v}^2}{f^2}} - 1 \right], \tag{7.65}$$

where

$$\mu_s = \frac{2\pi f \Delta z}{\hat{v}}, \tag{7.66}$$

and, in summary,

$$\text{total phase shift} = \mu = \mu_s + \mu_f. \tag{7.67}$$

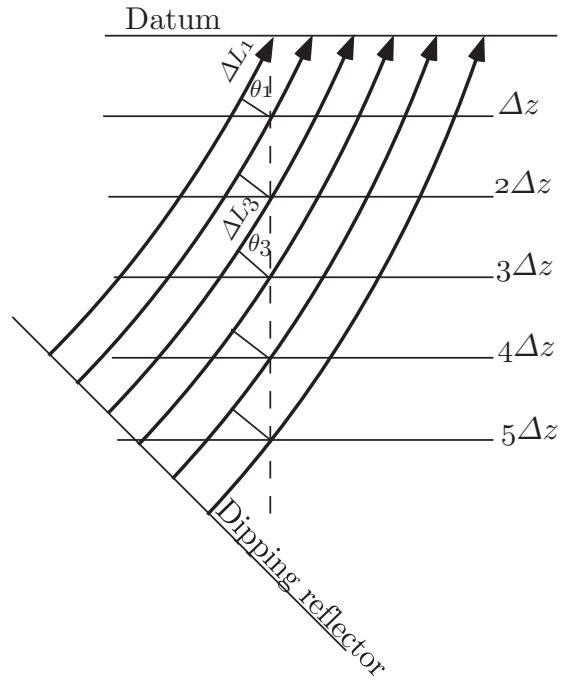


Figure 7.37 The recursive application of the constant-velocity phase shift can model $v(z)$ media as in Eq. (7.61).

The focusing phase shift is angle dependent and vanishes for $k_x = 0$. That the static phase shift accomplishes a bulk time delay can be seen as a consequence of the *phase shift theorem* of digital signal theory (e.g., Karl (1989), p. 87). This well-known result from time series analysis says that a time shift is accomplished in the Fourier domain by a phase shift, where the phase is a linear function of frequency. The slope of the linear phase function determines the magnitude of the time shift. Rather than merely quoting this result, it is instructive to actually demonstrate it. The static phase shift can be applied to the ERM seismogram by

$$\psi_s(x,t) = \int_{V_\infty} \phi_0(k_x,f) e^{-\mu_s + 2\pi i(k_x x - ft)}\, dk_x\, df. \tag{7.68}$$

Since the static phase shift is independent of k_x, the k_x integral can be done directly to give

$$\psi_s(x,t) = \int_{-\infty}^{\infty} \hat{\psi}_0(x,f) e^{-2\pi i(ft + f\Delta z/\hat{v})}\, df, \tag{7.69}$$

where $\hat{\psi}_0(x,f)$ is the temporal Fourier transform of the ERM seismogram. Letting $\tau = \Delta z/\hat{v}$, this becomes

$$\psi_s(x,t) = \int_{-\infty}^{\infty} \hat{\psi}_0(x,f) e^{-2\pi if(t+\tau)}\, df = \psi_0(x,t+\tau). \tag{7.70}$$

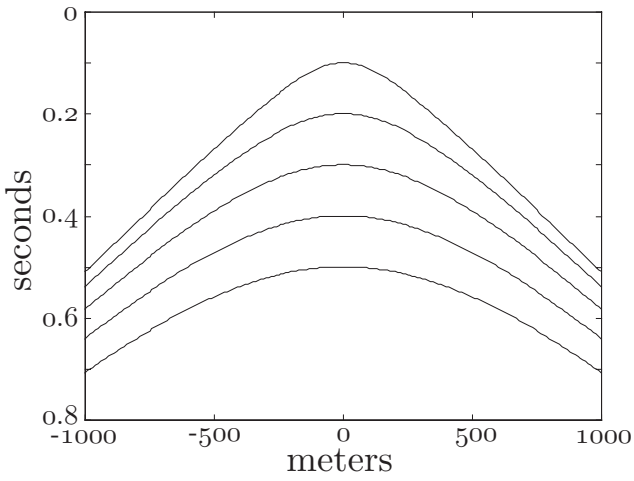


Figure 7.38 A diffraction chart showing the response of five point diffractors as recorded on the surface $z = 0$.

The last step follows because the integral is an inverse Fourier transform for the time coordinate $t + \tau$. This result is simply a proof of the phase shift theorem in the present context.

Wavefield Extrapolation in the Space–Time Domain

Since the extrapolation operator is applied with a multiplication in the Fourier domain, it must be a convolution in the space–time domain. In Section 7.2.4, constant-velocity migration was shown to be a nonstationary convolution. Constant-velocity extrapolation, which is a much simpler process than migration, is a stationary convolution.

Figure 7.38 shows the response of five different point diffractors, at depths of 200, 400, 600, 800, and 1000 m, as recorded at $z = 0$. This diffraction chart was constructed with an exploding reflector velocity $\hat{v} = 2000$ m/s. The chart represents an idealized ERM seismogram for a geology that consists of only five point diffractors. It is desired to determine the space–time shape of the extrapolation operator that will extrapolate this seismogram to 200 m. Since this is the depth of the first diffractor, the first diffraction curve should focus to a point and shift to time zero. The other diffraction curves should all focus somewhat and shift to earlier times by the same amount. In fact, the second hyperbola should be transformed into the first, the third into the second, and so on.

In Section 7.2.4, it was seen that replacing each point in the ERM seismogram by a wavefront circle (the *operator*) will focus all hyperbolas at once. With extrapolation, only the first hyperbola should focus and the operator (i.e., the replacement curve) should be the same for all points. Clearly, the operator needs to be concave ($\smile$) to focus the convex ($\frown$) diffraction curves. One process that will focus the first diffraction curve is to *crosscorrelate* the seismogram with the first diffraction curve itself. To visualize this, imagine tracing the first diffraction curve on a clear piece of film (being careful to mark the coordinate origin

on the film) and then placing the apex of this curve at some point on the diffraction chart. The value of the crosscorrelation is computed by taking the sample-by-sample product of the ERM seismogram and the section represented by the film, and then summing all of the resulting products. Assuming that all amplitudes are either zero (white) or one (black) for simplicity, this reduces to summing the samples in the ERM seismogram that are coincident in space–time with the diffraction curve on the film. This crosscorrelation value is then assigned to the location on the ERM seismogram of the coordinate origin on the film. Clearly, when the film is positioned such that its diffraction curve exactly overlies the first diffraction curve, a large value for the crosscorrelation will result. In fact, since this is the only exact match between the ERM seismogram and the curve on the film, this will be the largest value for the crosscorrelation. This exact match will occur when the coordinate origin on the film coincides with that on the ERM seismogram.

Conceptually, the process just described is very simple, though it might seem tedious to compute for all samples. Could this crosscorrelation process be the space–time equivalent of extrapolation? In fact, it is, and to realize this it helps to recall from time series analysis that the crosscorrelation of $a(t)$ with $b(t)$ can be done by time reversing $b(t)$ and convolving. That is, $a(t) \otimes b(t) = a(t) \bullet b(-t)$. This result carries over into two dimensions, so that the two-dimensional crosscorrelation of the ERM seismogram with the first diffraction curve can be done by reversing the curve in both space and time and then doing a two-dimensional convolution. In this case the diffraction curve is symmetric in space (it will not always be, though), so the reversal in space does nothing. However, the time reversal flips the first diffraction curve upside down to produce the concave operator that was envisioned.

Figures 7.39 and 7.40 illustrate these concepts with the extrapolation of the ERM seismogram of Figure 7.38 to the depth of the first point diffractor. The extrapolation operator is the time reverse of the first diffraction curve. In Figure 7.39A, only the focusing term is applied to the first diffraction curve, with the result that the curve is focused at its apex. In Figure 7.39B, the thin lens term is also included so that the focused diffraction curve is shifted to time zero. In Figure 7.40A, the focusing term is applied to the second diffraction curve and only partial focusing is achieved. When the thin lens term is included in Figure 7.40B, it is apparent that the partially focused second hyperbola is now equivalent to the first hyperbola.

This perspective of wavefield extrapolation is very powerful and general. The crosscorrelation argument shows that to extrapolate a wavefield to a particular depth, the response of a point diffractor at that depth as viewed from the current datum must be constructed. Then the wavefield is crosscorrelated with the diffraction response. The convolution argument is completely equivalent and graphically illustrates the effects of the two parts of the operator: focusing and static shift. It also shows how the extrapolation operator is similar to but simpler than the migration operator.

7.4.4 Time and Depth Migration by Phase Shift

Thus far, theory has been developed to construct a migrated depth section, $\psi(x, z, t = 0)$, from a zero-offset section or ERM seismogram. It is often desired to express the migrated

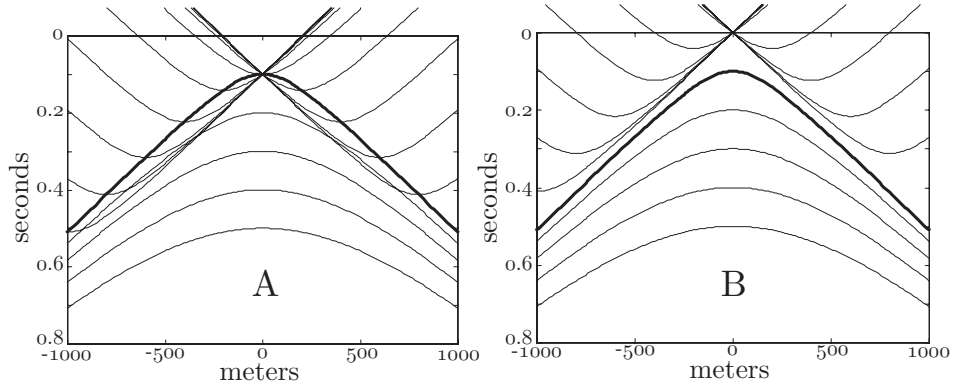


Figure 7.39 The first hyperbola of the diffraction chart of Figure 7.38 is shown convolved with its time reverse. (A) The focusing term only is applied. (B) Both the focusing and the thin lens term are applied.

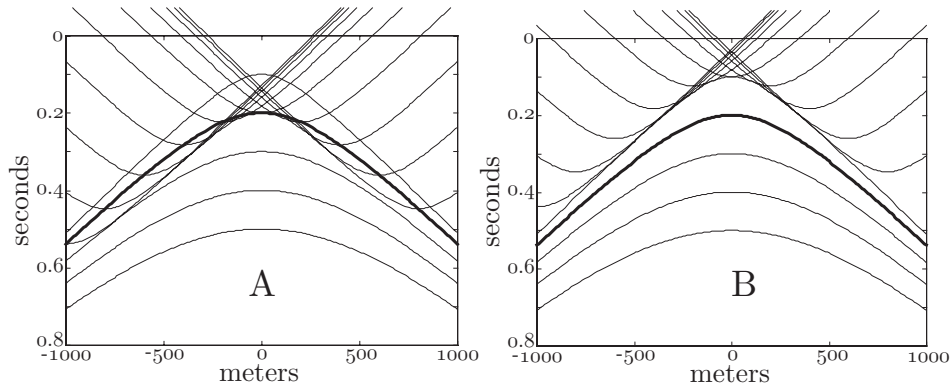


Figure 7.40 The second hyperbola of the diffraction chart of Figure 7.38 is shown convolved with the time reverse of the first hyperbola. (A) The focusing term only is applied. (B) Both the focusing and the thin lens term are applied.

data with a vertical time coordinate for interpretation. Called τ, this migrated time may be created from z by doing a simple one-dimensional stretch using the migration velocity model as discussed in Section 7.2.1. It is always correct to do this following migration with $v(x,z)$, with the stretch being defined by

$$\tau(x,z) = \int_0^z \frac{dz'}{\hat{v}(x,z')}. \tag{7.71}$$

A time display $\psi(x,\tau)$, created from a migrated depth section $\psi(x,z)$ using Eq. (7.71), is called a *migrated time* display. Generally, this has a meaning distinct from that of *time migration*. The two are equivalent only when lateral velocity variations are absent.

Time migration seeks to create $\psi(x,\tau)$ directly without first creating $\psi(x,z)$. To see how this might be done, recall that the extrapolation phase shift can be written in the *f–k*

domain as

$$\phi(\Delta z) = \phi_0 e^{i\mu_s + i\mu_f}, \tag{7.72}$$

where the static phase shift μ_s is given by Eq. (7.66) and the focusing phase shift is given by Eq. (7.65). If this extrapolator is applied in a recursive scheme, it is μ_s that progressively moves data to earlier times so that each depth sample can be estimated through the imaging condition $\psi(x, z, t = 0)$. If an extrapolation were done with only μ_f, then diffractions would focus at their apex time but never move to time zero, and flat events (i.e., $k_x = 0$) would never move at all. This is exactly the behavior desired of a time migration. In fact, a time migration can be computed by recursive phase shift using the extrapolation equation

$$\phi(\tau_m + \Delta\tau) = \phi(\tau_m) e^{i\mu_{fm}}, \tag{7.73}$$

where μ_{fm} is given by

$$\mu_{fm} = 2\pi\, \Delta\tau \left[\sqrt{1 - \frac{k_x^2 \hat{v}_m^2}{f^2}} - 1 \right], \tag{7.74}$$

and where $\Delta\tau = \Delta z / \hat{v}_m$ has been used. When computing a time migration by recursive extrapolation, the imaging condition must be changed because data no longer moves to $t = 0$ as it is focused. Instead, focused events are found at $t = \tau$, so that the time migration is given by $\phi(x, \tau, t = \tau)$.

Recursive extrapolation with Eq. (7.73), or some approximation to it, is a form of time migration, while recursive extrapolation with Eq. (7.72) is a depth migration. There is no essential difference between the two for $v(z)$, because the depth migration can be stretched to time with Eq. (7.71) or the time migration can be stretched to depth with

$$z(\tau) = \int_0^\tau \hat{v}(x, \tau')\, d\tau'. \tag{7.75}$$

To see this last point more clearly, if a recursive time migration is formulated as was done for depth migration that leads to Eq. (7.62), it will be seen that the time migration omits the accumulated phase shift operator

$$e^{i \int_0^z \mu_s(z')\, dz'} = \exp\left(i2\pi f \int_0^z \frac{dz'}{v(z')} \right). \tag{7.76}$$

A stretch of the time migration to depth effectively applies this operator. Time migration is thus equivalent to depth migration for $v(z)$ because the static delay, $\Delta z / \hat{v}(z) = \Delta\tau$, does not depend on x. Since data moves all around during migration following various raypaths, any accumulated time delays in $\psi(x, \tau, t = \tau)$ will be complicated functions of dip and position if $\hat{v}$ varies with x. When $\hat{v}$ varies only with z, then all data at fixed τ has the same accumulated delay regardless of migration raypath. When $\hat{v}$ varies laterally, time migration actually refracts data in a way that systematically violates Snell's law, as was discussed in Section 7.2.3.

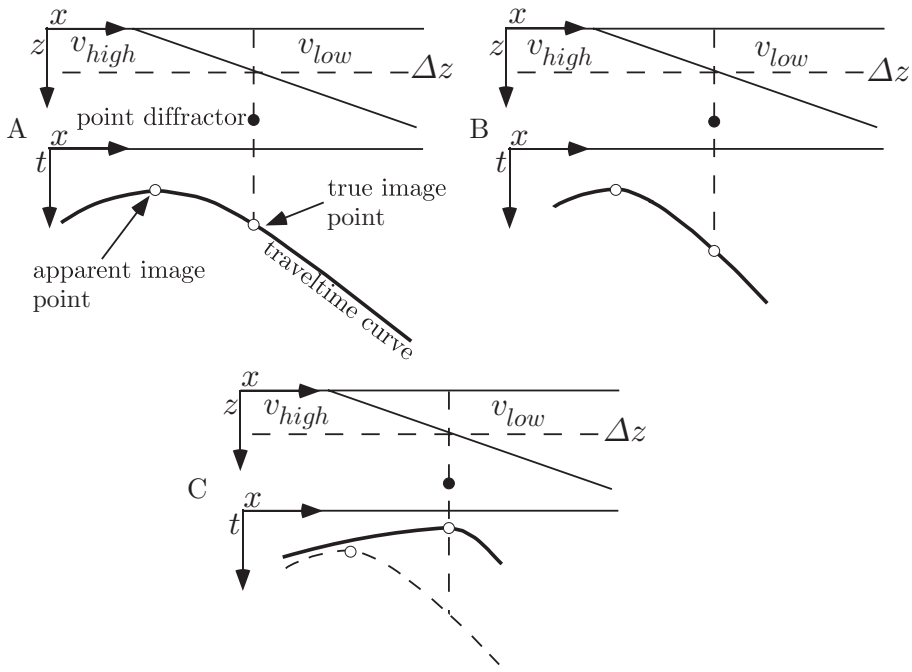


Figure 7.41 (A) A point diffractor sits beneath a low-velocity wedge (top) and its ERM seismogram shows an apparent image point displaced to the left (bottom). (B) After application of the focusing phase shift, the diffraction curve is partially focused but the apparent image point has not moved. (C) The static phase shift moves the apparent image point toward the true image point.

Figure 7.41A shows the ERM response of a point diffractor in a medium with $\hat{v}(x,z)$. The apparent image point, which is the crest of the traveltime curve, is displaced to the left because the velocity is higher there. In Figure 7.41B, the ERM seismogram has been extrapolated to half the depth of the diffractor using the focusing phase shift only. Since μ_{f} vanishes for $k_x = 0$, it cannot shift the crest of the traveltime curve even if it can apply the lateral velocity variations. All it achieves is a partial focusing of the diffraction response. In Figure 7.41C, the static phase shift has been applied and has had the effect of moving the crest of the traveltime curve to the right toward the true image point. This happens because the time advance of the static shift is greater for lower velocities than for higher ones. Thus it is apparent that extrapolation using μ_{s} and μ_{f} can focus the diffractor at the proper location, while μ_{f} alone can never do so.

7.5 Kirchhoff Methods

The Fourier methods discussed in Section 7.4 are based upon the solution of the scalar wave equation using Fourier transforms. This solution can be derived from a more

fundamental approach to partial differential equations called *separation of variables*. An alternative, and equally fundamental, approach to solving the same partial differential equations is based on *Green's theorem* and leads to the family of migration methods known as *Kirchhoff* methods. Though Kirchhoff methods seem superficially quite different from Fourier methods, the uniqueness theorems from partial differential equation theory guarantee that they are equivalent. However, this equivalence applies only to problems for which both approaches provide exact solutions to the wave equation, and that is the case for a constant-velocity medium with regular wavefield sampling and a horizontal recording surface. In all other cases, the two methods are implemented with differing approximations and can give very distinct results. Furthermore, even in the exact case, the methods have different computational artifacts.

The wavefront migration methods discussed in Section 7.2.4 are simplified examples of Kirchhoff migration. There are two alternative Kirchhoff approaches that differ in that one does the computations on the output space (x, z) and the other does them on the input space (x, t). In the first case, the method of replacing each point of the input section by a wavefront circle, as shown in Figure 7.11b, essentially works in the output space. Kirchhoff migration theory provides a detailed prescription for computing the amplitude and phase along the wavefront, and, in the case of variable velocity, the shape of the wavefront. The second case essentially sums through the input space along hyperbolic paths to compute each point in the output space. As shown in Figure 7.14, summation along hyperbolas and superposition of circular wavefronts lead to the same result. Again in this case, Kirchhoff theory shows that the summation along the hyperbolas must be done with specific weights and, for variable velocity, it shows how the hyperbola is replaced by a more general shape.

7.5.1 Gauss's Theorem and Green's Identities

Recall the fundamental theorem of calculus that says

$$\int_a^b \phi'(x)\,dx = \phi(x)\Big|_a^b = \phi(b) - \phi(a). \tag{7.77}$$

An interpretation of this statement is that the integral of a function ϕ' over the interval $a \le x \le b$ is found by evaluating a different function ϕ at the endpoints of the interval. The theorem assures us that if ϕ' exists, then so does ϕ under a fairly general set of conditions. These functions are, of course, mathematically related and ϕ is said to be the *integral* of ϕ' or, equivalently, ϕ' is the derivative of ϕ.

Gauss's theorem generalizes this basic result to dimensions greater than one. That is, it says that if the integral over a volume of a certain function is desired, then it can be obtained by evaluating another related function over the surface that bounds the volume. In higher dimensions, the notion of direction is more complicated than the $+$ or $-$ needed in one dimension, and vector functions express this. Gauss's theorem is usually written

$$\int_V \vec{\nabla} \cdot \vec{A}\,d\,\text{vol} = \oint_{\partial V} \vec{A} \cdot \vec{n}\,d\,\text{surf}. \tag{7.78}$$

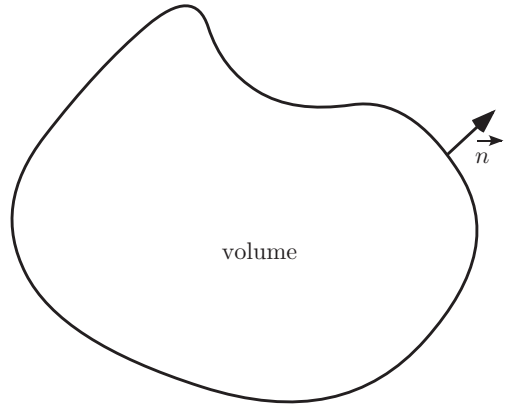


Figure 7.42 Gauss's theorem relates the volume integral of a function $\vec{\nabla} \cdot \vec{A}$ to the surface integral of a related function $\vec{A} \cdot \vec{n}$, where $\vec{n}$ is the outward-pointing normal to the surface ∂V bounding the integration volume V.

Here $\vec{A}$ is a vector function that might physically represent something like a fluid flow or electric field, and $\vec{n}$ is the *outward-pointing* normal to the surface bounding the volume of integration (Figure 7.42). Equation (7.78) generalizes Eq. (7.77) in several ways. First, a vector function $\vec{A}$ generalizes the notion of direction where, in one dimension, the sign of ϕ was sufficient. Second, the derivative has been generalized from ϕ' to $\vec{\nabla} \cdot \vec{A}$, and is called the divergence of $\vec{A}$. Finally, the dot product of $\vec{A}$ with $\vec{n}$ integrated over the bounding surface generalizes the simple difference $\phi(b) - \phi(a)$. In the one-dimensional case, the outward-pointing normal points in the $+x$ direction at b and in the $-x$ direction at a and the surface integral degenerates to a simple sum of two end members.

In many important cases, the vector function $\vec{A}$ can be calculated as the gradient of a scalar potential $\vec{A} = \vec{\nabla}\phi$. In this case Gauss's theorem becomes

$$\int_V \nabla^2 \phi \, d\,\mathrm{vol} = \oint_{\partial V} \frac{\partial \phi}{\partial n} \, d\,\mathrm{surf}, \tag{7.79}$$

where $\nabla^2 \phi = \vec{\nabla} \cdot \vec{\nabla}\phi$ and $\partial\phi/\partial n = \vec{\nabla}\phi \cdot \vec{n}$ have been used. The meaning of $\partial\phi/\partial n = \partial_n \phi$ is that it is the component of the vector $\vec{\nabla}\phi$ that is normal to the surface.

Now, we return to Eq. (7.77) and consider the case when $\phi = \phi_1 \phi_2$. Then $\phi' = \phi_2 \phi_1' + \phi_1 \phi_2'$ and

$$\int_a^b \left[\phi_2 \phi_1' + \phi_1 \phi_2'\right] dx = \phi_1 \phi_2 \Big|_a^b, \tag{7.80}$$

or

$$\int_a^b \phi_2 \phi_1' \, dx = \phi_1 \phi_2 \Big|_a^b - \int_a^b \phi_1 \phi_2' \, dx, \tag{7.81}$$

which is the formula for *integration by parts*. An analogous formula in higher dimensions arises by substituting $A = \phi_2 \vec{\nabla}\phi_1$ into Eq. (7.78). Using the identity $\vec{\nabla} \cdot a \vec{\nabla} b = \vec{\nabla} a \cdot \vec{\nabla} b +$

$a\,\nabla^2 b$ leads immediately to

$$\int_V \left[\vec{\nabla}\phi_2 \cdot \vec{\nabla}\phi_1 + \phi_2\,\nabla^2\phi_1\right] d\,\text{vol} = \oint_{\partial V} \phi_2 \frac{\partial \phi_1}{\partial n}\, d\,\text{surf}. \tag{7.82}$$

Then, letting $A = \phi_1\,\vec{\nabla}\phi_2$ leads to a similar result,

$$\int_V \left[\vec{\nabla}\phi_1 \cdot \vec{\nabla}\phi_2 + \phi_1\,\nabla^2\phi_2\right] d\,\text{vol} = \oint_{\partial V} \phi_1 \frac{\partial \phi_2}{\partial n}\, d\,\text{surf}. \tag{7.83}$$

Finally, subtracting Eq. (7.83) from (7.82) results in

$$\int_V \left[\phi_2\,\nabla^2\phi_1 - \phi_1\,\nabla^2\phi_2\right] d\,\text{vol} = \oint_{\partial V} \left[\phi_2 \frac{\partial \phi_1}{\partial n} - \phi_1 \frac{\partial \phi_2}{\partial n}\right] d\,\text{surf}, \tag{7.84}$$

which is known as *Green's second identity*.

Green's second identity is fundamental to the derivation of Kirchhoff migration theory. It is a multidimensional generalization of the integration-by-parts formula from elementary calculus and is valuable for its ability to solve certain partial differential equations. At this point, only geometric principles and vector calculus have been involved; the potential physical applications are as yet unspecified. That is, ϕ_1 and ϕ_2 in Eq. (7.84) are completely arbitrary scalar fields. They may be chosen as desired to conveniently express solutions to a given problem. Typically, in the solution to a partial differential equation such as the wave equation, one function is chosen to be the solution to the problem at hand and the other is chosen to be the solution to a simpler *reference problem*. The reference problem is usually selected to have a known analytic solution, and that solution is called a *Green's function*.

7.5.2 The Kirchhoff Diffraction Integral

Let ψ be a solution to the scalar wave equation

$$\nabla^2\psi(\vec{x},t) = \frac{1}{v^2}\frac{\partial^2\psi(\vec{x},t)}{\partial t^2}, \tag{7.85}$$

where the velocity v may depend upon position or may be constant. To eliminate the time dependence in Eq. (7.85), let ψ be given by a single Fourier component $\psi(\vec{x},t) = \hat{\psi}(\vec{x})e^{-2\pi i f t}$. Then Eq. (7.85) becomes the *Helmholtz equation*,

$$\nabla^2\hat{\psi}(\vec{x}) = -k^2\hat{\psi}(\vec{x}), \tag{7.86}$$

where $k^2 = 4\pi^2 f^2/v^2$. Now, let $g(\vec{x};\vec{x}_0)$ be the solution to

$$\nabla^2 g(\vec{x};\vec{x}_0) - k_0^2 g(\vec{x};\vec{x}_0) = \delta(\vec{x}-\vec{x}_0), \tag{7.87}$$

where $k_0^2 = 4\pi^2 f^2/v_0^2$, with v_0 constant over all space, and $\delta(\vec{x}-\vec{x}_0)$ represents a source at $\vec{x} = \vec{x}_0$. The analytic solution to Eq. (7.87) is well known (e.g., Morse and Feshbach

(1953), p. 810) and can be built from a linear combination of the two functions

$$g^{\pm}(\vec{x};\vec{x}_0) = \frac{e^{\pm ik_0 r}}{r}, \tag{7.88}$$

where $r = |\vec{x} - \vec{x}_0|$ and three spatial dimensions are assumed. In two dimensions, the solution must be expressed with Hankel functions that have the asymptotic form

$$g^{\pm}(\vec{x};\vec{x}_0) \sim \sqrt{\frac{2\pi}{kr}} e^{\pm ikr + i\pi/4}, \quad r \to \infty. \tag{7.89}$$

Since $g(\vec{x};\vec{x}_0)e^{-2\pi ift}$ is the time-dependent Green's function, it is apparent that $g^+ = r^{-1}e^{ik_0 r}$ corresponds to a wavefield traveling out from $r = 0$, while $g^- = r^{-1}e^{-ik_0 r}$ is a wavefield traveling inward toward $r = 0$. In modeling, g^+ is commonly used and is called the *causal Green's function*, while in migration it turns out that g^- is appropriate, and this is called the *anticausal Green's function*.

Now, we apply Green's identity (Eq. (7.84)) using $\hat{\psi}$ and g^- to get

$$\begin{aligned} &\int_V \left[g^-(\vec{x};\vec{x}_0)\, \nabla^2 \hat{\psi}(\vec{x}) - \hat{\psi}(\vec{x})\, \nabla^2 g^-(\vec{x};\vec{x}_0) \right] d\,\text{vol} \\ &\quad = \oint_{\partial V} \left[g^-(\vec{x};\vec{x}_0) \frac{\partial \hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x}) \frac{\partial g^-(\vec{x};\vec{x}_0)}{\partial n} \right] d\,\text{surf}. \end{aligned} \tag{7.90}$$

Substituting Eqs. (7.86) and (7.87) into the left-hand side of this expression leads to

$$\begin{aligned} &\int_V \left[k^2 - k_0^2 \right] g^-(\vec{x};\vec{x}_0)\, \hat{\psi}(\vec{x})\, d\,\text{vol} + \int_V \hat{\psi}(\vec{x})\, \delta(\vec{x} - \vec{x}_0)\, d\,\text{vol} \\ &\quad = \oint_{\partial V} \left[g^-(\vec{x};\vec{x}_0) \frac{\partial \hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x}) \frac{\partial g^-(\vec{x};\vec{x}_0)}{\partial n} \right] d\,\text{surf}. \end{aligned} \tag{7.91}$$

Assuming that the point $\vec{x}_0$ is interior to the volume V, the delta function collapses the second integral on the left and this expression can be rewritten as

$$\hat{\psi}(\vec{x}_0) = \Lambda(\vec{x}_0) + \oint_{\partial V} \left[g^-(\vec{x};\vec{x}_0) \frac{\partial \hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x}) \frac{\partial g^-(\vec{x};\vec{x}_0)}{\partial n} \right] d\,\text{surf}, \tag{7.92}$$

where

$$\Lambda(\vec{x}_0) \equiv \int_V \left[k^2 - k_0^2 \right] g^-(\vec{x};\vec{x}_0)\, \hat{\psi}(\vec{x})\, d\,\text{vol}. \tag{7.93}$$

Equation (7.92) estimates the wavefield $\hat{\psi}$ at the point $\vec{x}_0$ interior to V as a volume integral plus a surface integral over ∂V. The surface integral is what is desired, since we can hope to know $\hat{\psi}$ over the boundary of V. However, the volume integral involves the unknown $\hat{\psi}$ and is essentially not computable. The function $\Lambda(\vec{x}_0)$ expresses this volume integral and can be seen to vanish if the reference medium v_0 is equivalent to the actual medium v over the entire volume. Since g has been chosen as a constant-velocity Green's

function, Λ can only vanish precisely for constant velocity. However, in the variable-velocity case, approximate ray-theoretic Green's functions can be used to help minimize Λ (Docherty, 1991). To the extent that the reference medium does not equal the true medium, then Λ expresses the error in a $\hat{\psi}$ that is computed without Λ. In any case, the next step is to drop Λ and substitute $g^- = e^{-ik_0 r}/r$ into Eq. (7.92), with the result

$$\hat{\psi}(\vec{x}_0) = \oint_{\partial V} \left[\frac{e^{-ik_0 r}}{r} \frac{\partial \hat{\psi}(\vec{x})}{\partial n} - \hat{\psi}(\vec{x}) \frac{\partial}{\partial n} \frac{e^{-ik_0 r}}{r} \right] d\,\text{surf}. \tag{7.94}$$

The normal derivative of g can now be resolved into two terms,

$$\hat{\psi}(\vec{x}_0) = \oint_{\partial V} \left[\frac{e^{-ik_0 r}}{r} \frac{\partial \hat{\psi}(\vec{x})}{\partial n} + \frac{ik_0 \hat{\psi}(\vec{x}) e^{-ik_0 r}}{r} \frac{\partial r}{\partial n} + \hat{\psi}(\vec{x}) \frac{e^{-ik_0 r}}{r^2} \frac{\partial r}{\partial n} \right] d\,\text{surf}. \tag{7.95}$$

Multiplying both sides of this result by $e^{-2\pi i f t}$ and recalling that $\psi(\vec{x}_0, t) = \hat{\psi}(\vec{x}_0) e^{-2\pi i f t}$ gives

$$\psi(\vec{x}_0, t) = e^{-2\pi i f t} \oint_{\partial V} \left[\frac{e^{-ik_0 r}}{r} \frac{\partial \hat{\psi}(\vec{x})}{\partial n} + \frac{ik_0 \hat{\psi}(\vec{x}) e^{-ik_0 r}}{r} \frac{\partial r}{\partial n} + \hat{\psi}(\vec{x}) \frac{e^{-ik_0 r}}{r^2} \frac{\partial r}{\partial n} \right] d\,\text{surf}, \tag{7.96}$$

or, using $k_0 = 2\pi f / v_0$,

$$\psi(\vec{x}_0, t) = \oint_{\partial V} \frac{e^{-2\pi i f (t + r/v_0)}}{r} \left[\frac{\partial \hat{\psi}(\vec{x})}{\partial n} + \frac{i 2\pi f \hat{\psi}}{v_0} \frac{\partial r}{\partial n} + \hat{\psi}(\vec{x}) \frac{1}{r} \frac{\partial r}{\partial n} \right] d\,\text{surf}. \tag{7.97}$$

Now, $\hat{\psi}(\vec{x}) e^{-2\pi i f (t + r/v_0)} = \psi(\vec{x}, t + r/v_0)$ is the wavefield ψ at the point $\vec{x}$ but at the *advanced time* $t + r/v_0$. It is customary to denote this quantity by $[\psi]_{t+r/v_0}$, with the result

$$\psi(\vec{x}_0, t) = \oint_{\partial V} \left[\frac{1}{r} \left[\frac{\partial \psi}{\partial n} \right]_{t+r/v_0} - \frac{1}{v_0 r} \frac{\partial r}{\partial n} \left[\frac{\partial \psi}{\partial t} \right]_{t+r/v_0} + \frac{1}{r^2} \frac{\partial r}{\partial n} [\psi]_{t+r/v_0} \right] d\,\text{surf}, \tag{7.98}$$

where the time derivative in the second term results from $\partial_t \psi = -2\pi i f \psi$. This is a famous result and is known as Kirchhoff's diffraction integral. (In most textbooks this integral is derived for forward modeling, with the result that all of the terms are evaluated at the *retarded time* $t - r/v_0$ instead of the advanced time.) It expresses the wavefield at the observation point $\vec{x}_0$ at time t in terms of the wavefield on the boundary ∂V at the advanced time $t + r/v_0$. As with Fourier theory, it appears that knowledge of both ψ and $\partial_n \psi$ is necessary to reconstruct the wavefield at an internal point.

7.5.3 The Kirchhoff Migration Integral

There are two essential tasks required to convert Eq. (7.98) into a practical migration formula. First, as mentioned above, the apparent need to know $\partial_n \psi$ must be addressed. Second, the requirement that the integration surface must extend all the way around the

volume containing the observation point must be dropped. There are various arguments to deal with both of these points that have appeared in the literature. Schneider (1978) dispensed with the need to know $\partial_n \psi$ by using a dipole Green's function with an image source above the recording plane. The result was a Green's function that vanished at $z = 0$ and cancelled the $\partial_n \psi$ term in Eq. (7.98). Schneider also argued that the boundary surface can be a horizontal plane with a hemisphere below, and when the hemisphere is extended to infinity, contributions from it vanish. Wiggins (1984) adapted Schneider's technique to rough topography. Docherty (1991) showed that a monopole Green's function, as used here in Eq. (7.88), can lead to the accepted result and also challenged Schneider's argument that the integral over the infinite hemisphere can be neglected. Instead, Docherty formulated the expression with the *backscattered* wavefield received at the surface and simply put the lower part of the integration surface beneath the reflector. On physical grounds, the wavefield beneath the reflector is not expected to contain significant information about the backscattered field, and so it may be neglected. In reality, it does contain some information because the reflected and transmitted wavefields are related through boundary conditions on the reflector, but these are subtle second-order effects. It should also be recalled that virtually all authors on this subject have approached it with knowledge of the desired result. After all, migration by summation along diffraction curves or by wavefront superposition has been done for many years. Though Schneider's derivation has been criticized, his final expressions are considered correct.

As a first step in adapting Eq. (7.98), it is usually considered appropriate to discard the term $(1/r^2)(\partial r/\partial n)\,[\psi(\vec{x})]_{t+r/v_0}$. This is called the *near-field term* and decays more strongly with r than the other two terms. Then, the surface $S = \partial V$ is taken as the $z = 0$ plane, S_0, plus the surface infinitesimally below the reflector, S_z, and finally these surfaces are joined at infinity by vertical cylindrical walls, S_∞ (Figure 7.43). As mentioned previously, the integration over S_z is not expected to contribute significantly to reconstruction of the backscattered field. Also, the integration over S_∞, though it may contribute, can never be realized owing to finite-aperture limitations, and its neglect may introduce unavoidable artifacts. With these considerations, Eq. (7.98) becomes

$$\psi(\vec{x}_0, t) = \oint_{S_0} \left[\frac{-1}{r} \left[\frac{\partial \psi}{\partial z} \right]_{t+r/v_0} + \frac{1}{v_0 r} \frac{\partial r}{\partial z} \left[\frac{\partial \psi}{\partial t} \right]_{t+r/v_0} \right] d\,\text{surf}, \tag{7.99}$$

where the signs on the terms arise because $\vec{n}$ is the outward normal and z is increasing downward so that $\partial_n = -\partial_z$.

Now, $\partial_z \psi$ must be evaluated. Figure 7.43 shows the source wavefield being scattered from the reflector at $\vec{x}_0$, which is called the *scatterpoint*. A simple model for ψ is that it is approximately the wavefield from a point source, placed at the image source location, that passes through the scatterpoint to the receiver. This can be expressed as

$$\psi(\vec{x}, t) \sim \frac{1}{r} A\left(t - \frac{r}{v}\right) = \frac{[A]_{t-r/v}}{r}, \tag{7.100}$$

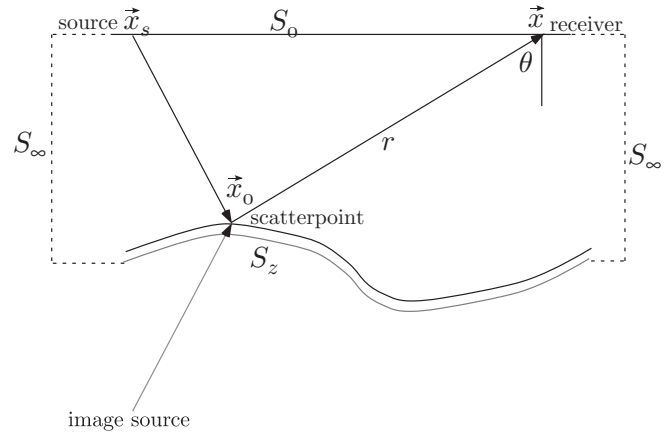


Figure 7.43 The geometry for Kirchhoff migration. The integration surface is $S_0 + S_z + S_\infty$ and it is argued that only S_0 contributes meaningfully to the estimation of the backscattered field at $\vec{x}_0$.

where $A(t)$ is the source waveform at the scatterpoint. Using the chain rule gives

$$\frac{\partial \psi}{\partial z} = \frac{\partial r}{\partial z}\frac{\partial \psi}{\partial r} = \frac{\partial r}{\partial z}\left[\frac{-1}{vr}\left[\frac{\partial A}{\partial t}\right]_{t-r/v} - \frac{[A]_{t-r/v}}{r^2}\right]. \tag{7.101}$$

If the second term (a near-field term) is neglected, this becomes

$$\frac{\partial \psi}{\partial z} = \frac{\partial r}{\partial z}\frac{-1}{vr}\left[\frac{\partial A}{\partial t}\right]_{t-r/v} = -\frac{\partial r}{\partial z}\frac{1}{v}\frac{\partial \psi}{\partial t}. \tag{7.102}$$

When this is substituted into Eq. (7.99), the two terms in square brackets become similar, both involving the time derivative of the advanced wavefield. These will combine if v_0 is now taken to be the same as v. Thus

$$\psi(\vec{x}_0, t) = \oint_{S_0} \frac{2}{vr}\frac{\partial r}{\partial z}\left[\frac{\partial \psi}{\partial t}\right]_{t+r/v} d\,\text{surf}. \tag{7.103}$$

Finally, consider $\partial_z r$. Since $r = \sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}$, this can be written as

$$\frac{\partial r}{\partial z} = \frac{\partial}{\partial z}\sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2} = \frac{z}{r} = \cos\theta, \tag{7.104}$$

where θ is the vertical angle between the receiver location and the ray to the scatterpoint. With this, the final formula for the scattered wavefield just above the reflector is

$$\psi(\vec{x}_0, t) = \oint_{S_0} \frac{2\cos\theta}{vr}\left[\frac{\partial \psi}{\partial t}\right]_{t+r/v} d\,\text{surf}. \tag{7.105}$$

Equation (7.105) is not yet a migration equation. As mentioned, it provides an estimate of the scattered wavefield just above the scatterpoint. Thus it is a form of wavefield

extrapolation, though it is direct, not recursive. A migration equation must purport to estimate reflectivity, not just the scattered wavefield, and for this purpose a model relating the wavefield to the reflectivity is required. The simplest such model is the exploding reflector model (Section 7.2.6), which asserts that the reflectivity is identical to the downward continued scattered wavefield at $t = 0$ provided that the downward continuation is done with $\hat{v} = v/2$. Thus, an ERM migration equation follows immediately from Eq. (7.105) as

$$\psi(\vec{x}_0, 0) = \oint_{S_0} \frac{2\cos\theta}{\hat{v}r} \left[\frac{\partial\psi}{\partial t}\right]_{r/\hat{v}} d\,\text{surf} = \oint_{S_0} \frac{4\cos\theta}{vr} \left[\frac{\partial\psi}{\partial t}\right]_{2r/v} d\,\text{surf}. \tag{7.106}$$

This result, derived by many authors, including Schneider (1978) and Scales (1995), expresses migration by summation along hyperbolic travelpaths through the input data space. The hyperbolic summation is somewhat hidden by the notation but is indicated by $[\partial_t\psi]_{2r/v}$. Recall that this notation means that the expression in square brackets is to be evaluated at the time indicated by the subscript. That is, as $\partial_t\psi(\vec{x}, t)$ is integrated over the $z = 0$ plane, only those specific traveltimes values are selected that obey

$$t = \frac{2r}{v} = \frac{2\sqrt{(x - x_0)^2 + (y - y_0)^2 + z_0^2}}{v}, \tag{7.107}$$

which is the equation of a zero-offset diffraction hyperbola. When squared, this result is a three-dimensional version of Eq. (7.36).

In addition to diffraction summation, Eq. (7.106) requires that the data be scaled by $4\cos\theta/(vr)$ and that the time derivative be taken before summation. These additional details were not indicated by the simple geometric theory of Section 7.2.4, and are major benefits of Kirchhoff theory. It is these sorts of corrections that are necessary to move toward the goal of creating band-limited reflectivity. The same correction procedures are contained implicitly in *f–k* migration.

The considerations taken into account in deriving Eq. (7.106) suggest why ERM migration does not achieve correct amplitudes. As mentioned following Eq. (7.105), a model linking the backscattered wavefield to reflectivity was required. A more physical model will illustrate the shortcomings of the exploding reflector model. Such a model has been advanced by Berkhout (1985) and others. They consider that the source wavefield propagates directly to the reflector, undergoing transmission losses and geometrical spreading but without multiples and converted modes. At the reflector, it is scattered upward with an angle-dependent reflection coefficient and then propagates upward to the receivers, with further geometrical spreading and transmission losses but again without multiples and mode conversions. This allows an interpretation of Eq. (7.105) as equivalent to the wavefield propagated from the source to the scatterpoint and scaled by the reflection coefficient. Thus, a better way to estimate the reflectivity is to produce a model of the source wavefield as propagated down to the scatterpoint and divide the result of Eq. (7.105) by this modeled wavefield. This is a very general imaging condition, called the *deconvolution imaging condition*, that works for prestack and zero-offset data. The ERM imaging condition is kinematically correct but does not achieve the same amplitudes as the deconvolution imaging condition.

Kirchhoff migration is one of the most adaptable migration schemes available. It can be easily modified to account for such difficulties as topography, irregular recording geometry, prestack migration, and converted wave imaging. When formulated as a depth migration, it tends to be a slow method because great care must be taken in the ray tracing (Gray, 1986). When formulated as a time migration, straight-ray calculations using rms velocities can be done to greatly speed the process. Another advantage of Kirchhoff methods is the ability to perform "target-oriented" migrations. That is, Eq. (7.106) need only be evaluated for those points in (x, z) which comprise the target. Since the cost of computation is directly proportional to the number of output points, this can greatly reduce the run times and makes migration parameter testing very feasible.

7.6 Finite-Difference Methods

7.6.1 Finite-Difference Extrapolation by Taylor Series

Finite-difference techniques are perhaps the most direct, but often the least intuitive, of migration methods. As the name implies, they involve the approximation of analytic derivatives of the wave equation with finite-difference expressions. To see how this might be done, consider the definition of the derivative of an arbitrary function $\psi(z)$,

$$\frac{d\psi(z)}{dz} = \lim_{\Delta z \to 0} \frac{\psi(z + \Delta z) - \psi(z)}{\Delta z}. \tag{7.108}$$

A simple finite-difference approximation for this derivative simply involves omitting the limit:

$$\frac{d\psi(z)}{dz} \approx \delta_z^{1+}\psi(z) \equiv \frac{\psi(z + \Delta z) - \psi(z)}{\Delta z}. \tag{7.109}$$

Here, δ_z^{1+} is the *first-order forward difference operator*; Δz is assumed small with respect to length scales of interest (i.e., wavelengths) but is still finite.

Finite-difference operators can be used to predict or extrapolate a function. Suppose that values for $\psi(z)$ and its first derivative are known at z; then Eq. (7.109) can be rearranged to predict $\psi(z + \Delta z)$:

$$\psi(z + \Delta z) \approx \psi(z) + \frac{d\psi(z)}{dz}\Delta z. \tag{7.110}$$

This can be regarded as a truncated Taylor series. Taylor's theorem provides a method for extrapolation of a function provided that the function and all of its derivatives are known at a single point:

$$\psi(z + \Delta z) = \psi(z) + \frac{d^1\psi(z)}{dz^1}\Delta z + \frac{1}{2}\frac{d^2\psi(z)}{dz^2}\Delta z^2 + \frac{1}{6}\frac{d^3\psi(z)}{dz^3}\Delta z^3 + \ldots. \tag{7.111}$$

If all derivatives exist up to infinite order at z, then there is no limit to the extrapolation distance. Essentially, if the function and all its derivatives are known at one location, then it is determined everywhere.

If finite-difference extrapolation is equivalent to using a truncated Taylor series, then what relation does wavefield extrapolation by phase shift have with Taylor series? Recall that if $\phi(z)$ is a solution to the scalar wave equation in the Fourier domain, then it can be extrapolated by

$$\phi(z + \Delta z) = \phi(z) e^{2\pi i k_z \Delta z}. \tag{7.112}$$

Here, k_z is the vertical wavenumber, whose value may be determined from the scalar wave dispersion relation, $k_z = \sqrt{f^2/v^2 - k_x^2}$. Furthermore, if $\phi_0(k_x, f)$ is the f–k spectrum of data measured at $z = 0$, then, for constant velocity, $\phi(z)$ is given by

$$\phi(z) = \phi_0 e^{2\pi i k_z z}. \tag{7.113}$$

As an illustration of the connection between Taylor series and the phase-shift extrapolator, consider using Taylor series applied to Eq. (7.113) to derive something equivalent to Eq. (7.112). From Eq. (7.113), the various derivatives can be estimated with the result that the nth derivative is

$$\frac{d^n \phi(z)}{dz^n} = [2\pi i k_z]^n \phi(z). \tag{7.114}$$

Then, using this result, a Taylor series expression for the extrapolation of $\phi(z)$ to $\phi(z + \Delta z)$ is

$$\phi(z + \Delta z) = \phi(z) + [2\pi i k_z]\phi(z)\,\Delta z + \frac{1}{2}[2\pi i k_z]^2 \phi(z)\,\Delta z^2 + \frac{1}{6}[2\pi i k_z]^3 \phi(z)\,\Delta z^3 + \ldots, \tag{7.115}$$

or

$$\phi(z + \Delta z) = \phi(z)\left[1 + 2\pi i k_z\,\Delta z + \frac{1}{2}[2\pi i k_z\,\Delta z]^2 + \frac{1}{6}[2\pi i k_z\,\Delta z]^3 + \ldots\right]. \tag{7.116}$$

At this point, recall that the expression for the series expansion of an exponential is $e^x = 1 + x + x^2/2 + x^3/6 + \ldots$, which affords the conclusion that the infinite series in square brackets in Eq. (7.116) may be summed to obtain $[1 + 2\pi i k_z\,\Delta z + \frac{1}{2}[2\pi i k_z\,\Delta z]^2 + \frac{1}{6}[2\pi i k_z\,\Delta z]^3 + \ldots] = e^{2\pi i k_z \Delta z}$. Thus, if infinitely many terms are retained in the series, Eq. (7.116) is equivalent to Eq. (7.112). It may be concluded that wavefield extrapolation by phase shift is equivalent to extrapolation with an infinite-order Taylor series and there is no upper limit on the allowed size of Δz (in constant velocity). Alternatively, wavefield extrapolation by finite-difference approximations is equivalent to extrapolation with a truncated Taylor series and the step size Δz will have a definite upper limit of validity.

7.6.2 Other Finite-Difference Operators

The finite-difference approximation used in Eq. (7.109) is the simplest possible, and there are many others. The subject of solving partial differential equations by finite-difference methods has produced a vast literature, but only a brief examination is possible here. Aki and Richards (1980), Ames (1992), and Durran (1999) contain much more thorough discussions.

The forward difference operator in (7.109) uses the points at z and $z + \Delta z$. A *backward difference operator* is equally acceptable as an approximate derivative:

$$\frac{d\psi(z)}{dz} \approx \delta_z^{1-}\psi(z) \equiv \frac{\psi(z) - \psi(z - \Delta z)}{\Delta z}. \tag{7.117}$$

Both of these expressions give similar problems in practice that are related to the fact that the difference is not centered at the estimation point. That is, $\psi(z + \Delta z) - \psi(z)$ is centered at $z + \Delta z/2$, while the backward difference is centered at $z - \Delta z/2$. This suggests forming a *centered difference* by averaging the forward and backward operators:

$$\frac{d\psi(z)}{dz} \approx \frac{1}{2}\left[\delta_z^{1+} + \delta_z^{1-}\right]\psi(z) \equiv \frac{\psi(z + dz) - \psi(z - \Delta z)}{2\,\Delta z} \equiv \delta_z^{1}. \tag{7.118}$$

The centered difference is usually superior to the forward and backward differences and should be used whenever possible.

An approximation for the second derivative can be developed by applying the forward and backward operators in succession:

$$\begin{aligned}\frac{d^2\psi(z)}{dz^2} &\approx \delta_z^{1+}\delta_z^{1-}\psi(z) = \delta_z^{1+}\left[\frac{\psi(z) - \psi(z - \Delta z)}{\Delta z}\right] \\ &= \frac{\psi(z + \Delta z) - \psi(z)}{\Delta z^2} - \frac{\psi(z) - \psi(z - \Delta z)}{\Delta z^2},\end{aligned} \tag{7.119}$$

or

$$\frac{d^2\psi(z)}{dz^2} \approx \delta_z^2\psi(z) \equiv \frac{\psi(z + \Delta z) - 2\psi(z) - \psi(z - \Delta z)}{\Delta z^2}n. \tag{7.120}$$

This is a centered approximation to the second derivative.

7.6.3 Finite-Difference Migration

Most finite-difference migration methods are formulated in the space–time domain, though space–frequency methods are also common. Also, they are all recursive extrapolation techniques. Though there are many equivalent ways to develop any particular method, a consistent approach can be formulated by beginning in the frequency domain with phase shift theory as follows:

1. Develop a suitable rational approximation to the one-way dispersion relation for scalar waves. A rational approximation is one which eliminates the square root, though it may involve multiplication, division, and powers of frequencies and wavenumbers.
2. Construct a space–time-domain differential equation from the approximate dispersion relation by the replacement rules

$$f \to \frac{i}{2\pi}\frac{\partial}{\partial t}, \quad k_x \to \frac{-i}{2\pi}\frac{\partial}{\partial x}, \quad k_y \to \frac{-i}{2\pi}\frac{\partial}{\partial y}, \quad k_z \to \frac{-i}{2\pi}\frac{\partial}{\partial z}. \tag{7.121}$$

3. Choose an appropriate form for the finite-difference operators (i.e., forward, backward, or central differences).

4. Develop the difference equation which corresponds to the differential equation found in step 2.
5. Solve the difference equation for the extrapolated wavefield.
6. Implement a migration by a recursive extrapolation computer algorithm using step 5.

As an illustration, the *15 degree* finite-difference time migration algorithm will be developed. This was one of the first migration algorithms to be developed and is described in Claerbout (1976). Recall that in the Fourier domain, time migration by recursive extrapolation proceeds by using the focusing phase shift at each extrapolation step,

$$\mu_{\mathrm{f}} = \frac{2\pi f \Delta z}{\hat{v}} \left[\sqrt{1 - \frac{k_x^2 \hat{v}^2}{f^2}} - 1 \right]. \tag{7.122}$$

Since the general form of an extrapolation phase shift is phase $= 2\pi k_z \, \Delta z$, a finite-difference time migration requires a "wave equation" whose dispersion relation approximates

$$\tilde{k}_z = \frac{f}{\hat{v}} \left[\sqrt{1 - \frac{k_x^2 \hat{v}^2}{f^2}} - 1 \right], \tag{7.123}$$

where $\tilde{k}_z$ is used rather than k_z to reserve the latter term for the real vertical wavenumber. It is not acceptable to perform the substitutions of Eq. (7.121) into this result, because the square root of a differential operator results and that is difficult to work with. Therefore, a rational approximation to the square root in Eq. (7.123) is desired. The second term in the square root is $\sin^2 \theta$, where θ is the scattering angle. For small angles (i.e., energy traveling nearly vertically), it is expected that an approximate form can be obtained by expanding the square root and keeping only the first few terms:

$$\sqrt{1 - \frac{k_x^2 \hat{v}^2}{f^2}} = 1 - \frac{k_x^2 \hat{v}^2}{2f^2} + \frac{3k_x^2 \hat{v}^2}{8f^2} + \ldots. \tag{7.124}$$

Truncating Eq. (7.124) at two terms and substituting the result into Eq. (7.123) gives

$$\tilde{k}_z \approx \frac{f}{\hat{v}} \left[1 - \frac{k_x^2 \hat{v}^2}{2f^2} - 1 \right] = -\frac{k_x^2 \hat{v}^2}{2f}. \tag{7.125}$$

To construct a partial differential equation from this approximate dispersion, we first multiply both sides by $2f/\hat{v}$ and replace the spectral multiplications with partial derivatives according to Eq. (7.121). This results in

$$\frac{2}{\hat{v}} \frac{\partial^2 \psi(x, \tilde{z}, t)}{\partial \tilde{z} \, \partial t} + \frac{\partial^2 \psi(x, \tilde{z}, t)}{\partial x^2} = 0. \tag{7.126}$$

This is known as the *15 degree* or *parabolic* wave equation. The reason for this name is suggested by Figure 7.44, which shows that Eq. (7.125) is a good approximation to Eq. (7.123) for angles less than about 15°.

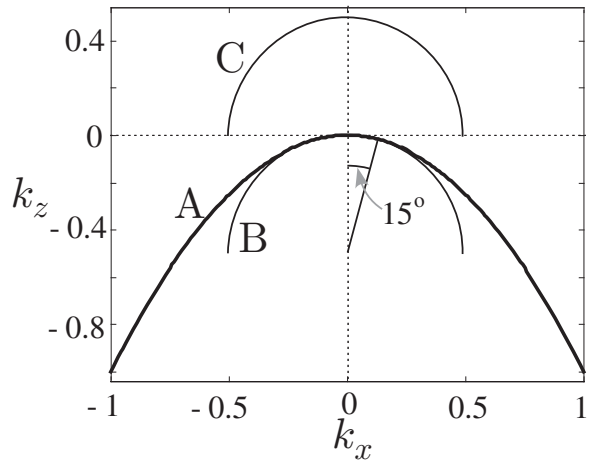


Figure 7.44 For a particular f and $\hat{v}$, (A) the dispersion relation (Eq. (7.125)) of the parabolic wave equation, (B) the exact dispersion relation (Eq. (7.123)) of the focusing phase shift, and (C) the full dispersion relation of the scalar wave equation.

Before proceeding with the finite-difference approximations, it is helpful to change variables in Eq. (7.126) from $\tilde{z}$ to τ, where $\tau = \tilde{z}/\hat{v}$. The $\tilde{z}$ derivative changes according to $\partial_{\tilde{z}} = (\partial_{\tilde{z}}\tau)\partial_\tau = \hat{v}^{-1}\partial_\tau$. This allows Eq. (7.126) to be recast as

$$\frac{2}{\hat{v}^2}\frac{\partial^2\psi(x,\tau,t)}{\partial\tau\,\partial t} + \frac{\partial^2\psi(x,\tau,t)}{\partial x^2} = 0. \tag{7.127}$$

Equation (7.127) is an approximate wave equation which can be used to migrate stacked data from zero-offset time to migrated time. Though this equation is rarely used anymore, its finite-difference implementation illustrates most of the features of the method and is simpler than higher-order methods.

The finite-difference approximation to Eq. (7.127) is well described by Claerbout (1976), and that approach is followed here. Let

$$\frac{1}{\Delta x^2}T = \delta_x^2 \approx \frac{\partial^2}{\partial x^2} \quad \text{and} \quad \psi_j^k = \psi(x, k\,\Delta\tau, j\,\Delta t). \tag{7.128}$$

Then the x derivatives of Eq. (7.127) are approximated as

$$\delta_\tau\delta_t\psi_j^k = -\frac{\hat{v}^2}{2\,\Delta x^2}T\psi_j^k, \tag{7.129}$$

where δ_τ and δ_t are, as yet, unspecified finite-difference operators. The δ_t operator is implemented as a forward difference, but the right-hand side of Eq. (7.129) is also modified to ensure that both sides of the equation are centered at the same grid location. The result is

$$\delta_\tau\left[\psi_{j+1}^k - \psi_j^k\right] = -\frac{\Delta t\,\hat{v}^2}{4\,\Delta x^2}T\left[\psi_{j+1}^k + \psi_j^k\right], \tag{7.130}$$

where the right-hand side represents the average of T applied to two grid points. This process of maintaining grid balance is essential in producing a stable algorithm and is a *Crank–Nicholson* method. Next the τ operator is implemented in a similar way as a balanced forward difference, with the result

$$\left[\psi_{j+1}^{k+1} - \psi_{j+1}^{k}\right] - \left[\psi_{j}^{k+1} - \psi_{j}^{k}\right] = -\frac{\Delta\tau\ \Delta t\,\hat{v}^2}{8\ \Delta x^2} T\left[\psi_{j+1}^{k+1} + \psi_{j+1}^{k} + \psi_{j}^{k+1} + \psi_{j}^{k}\right]. \quad (7.131)$$

Finally, to continue downward, Eq. (7.131) must be solved for ψ_j^{k+1}. This can be written as

$$[I - aT]\,\psi_j^{k+1} = [I + aT]\left[\psi_{j+1}^{k+1} + \psi_j^k\right] - [I - aT]\,\psi_{j+1}^{k}, \quad (7.132)$$

where $a = \Delta\tau\ \Delta t\,\hat{v}^2/8\ \Delta x^2$. Equation (7.132) is solved numerically for the unknown ψ_j^{k+1}, where it is assumed that all of the quantities on the right-hand side are known. This is an example of an *implicit* finite-difference method because it requires the numerical inversion of the matrix operator $I + aT$.

The differencing procedure can be viewed on a (j, k) grid as shown in Figure 7.45. The x axis is orthogonal to this figure and is not shown. The first row contains the measured CMP stack and the last column (corresponding to maximum time) is set to zero. Consider the four grid points in the upper right corner. Three of these are prescribed by initial conditions and the fourth, ψ_{n-1}^1, can then be calculated by Eq. (7.132). Subsequently, ψ_{n-2}^1 can be solved for and so on until the entire second row is calculated. Computation then moves to the third row and continues until the entire grid is completed. Each row corresponds to an extrapolated τ section and the final migrated section corresponds to $t = \tau$, which is the diagonal.

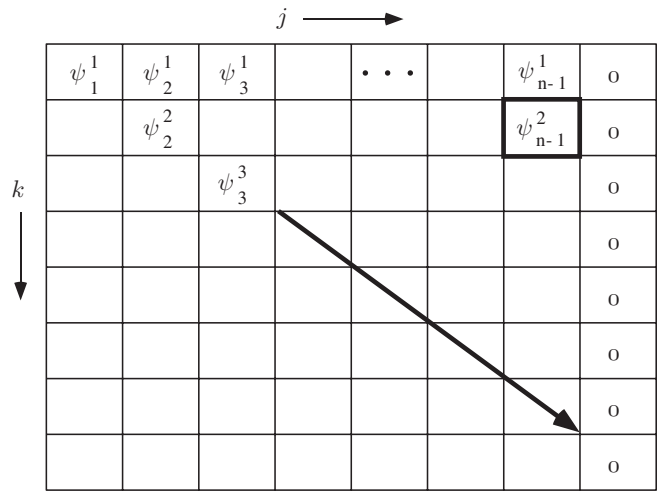


Figure 7.45 The parabolic wave equation is solved on a 2D grid using finite differences. The first row contains the known data and the last column is set to zero. The solution proceeds row by row and begins in the upper right corner.

From this example, many of the well-known characteristics of finite-difference migration algorithms are easily deduced, such as:

- They are recursive extrapolation schemes.
- They use an approximate form of the exact dispersion relation for scalar waves. This means that they are all dip or scattering-angle limited.
- They are very flexible in handling velocity variations, since the velocity may simply be varied in each grid cell.
- The use of finite-difference derivatives means that even the approximate dispersion relation is not exactly realized.
- Finite-difference derivatives are not unique, and the accuracy of a method depends strongly on the sophistication of the approximations used. Generally, finite-difference methods need many more samples per wavelength than Fourier methods (6–10 versus 2–3).
- They can be either time or depth migrations.
- They can be posed in the space–time or space–frequency domain.

7.7 Practical Considerations for Finite Datasets

From the perspective of *f–k* migration theory (Section 7.4.1), optimizing the ability of seismic data to resolve Earth features requires maximization of the spectral bandwidth after migration. The k_x (horizontal wavenumber) bandwidth determines the lateral resolution and, as will be seen, is directly proportional to the maximum signal frequency and the sine of the maximum scattering angle and inversely proportional to velocity. Lateral resolution increases as the k_x bandwidth increases, so this means that increasing signal frequency bandwidth improves lateral resolution. The link with scattering angle means that the lateral resolution depends on the degree to which a seismic survey captures scattered energy from any point in the subsurface. It will soon be apparent that the ability to capture scattered energy is a strong function of position and is controlled primarily by aperture (related to seismic line length) and temporal record length. A further factor is that the spatial sampling must be sufficiently dense that all scattered energy is recorded without spatial aliasing.

Constraints on the maximum scattering angle can be derived by examining the three effects of finite spatial aperture, finite recording time, and discrete spatial sampling. Here, these effects are analyzed, assuming zero-offset recording, for the case of constant velocity and for a linear (constant-gradient) $v(z)$ medium. Explicit analytic expressions are derived for the limits imposed on scattering angle for each of the three effects. Plotting these scattering-angle limits versus depth limits for assumed recording parameters is an effective way to appreciate their impact on recording. When considered in context with *f–k* migration theory, these scattering-angle limits can be seen to limit spatial resolution and the possibility of recording specific reflector dips. Seismic surveys designed with the linear $v(z)$ theory are often much less expensive than constant-velocity-theory designs.

The seismic line length (more correctly, spatial aperture) and maximum record time place definite limits on the maximum scattering angle that can be recorded, and hence imaged, on a migrated zero-offset section. Since horizontal resolution depends directly on the sine of the maximum scattering angle (see Vermeer (1990) and many others), it is important to understand these effects for survey design and interpretation. Furthermore, the observation of a normal-incidence reflection from a dipping reflector requires having a scattering-angle spectrum whose limits exceed the reflector dip.

The imposition of finite recording apertures in space and time actually imprints a strong spatial–temporal variation (i.e., nonstationarity) on the maximum scattering angle and hence on the spectral content of a migrated section. As an example, consider the synthetic seismic section shown in Figure 7.46A. This shows a zero-offset (constant-velocity) simulation of the response of a grid of point scatterers (diffractors) distributed uniformly throughout the section. Note how the diffraction responses change shape with depth and how the recording apertures truncate each one differently. Figure 7.46B is a display of the *f–k* amplitude spectrum for this section. As expected from elementary theory, all energy is confined to a triangular region defined by $|k_x| < f/\hat{v}$.

Figure 7.47A shows this section after a constant-velocity *f–k* migration, and Figure 7.47B shows the *f–k* spectrum after migration. The spectrum shows the expected behavior in that the triangular region of Figure 7.46B has unfolded into a circle. Essentially, each frequency (horizontal line in Figure 7.46B) maps to a circle in Figure 7.47B (see Section 7.4.1 and Chun and Jacewitz (1981)). Note that *f–k* migration theory as usually stated (Stolt, 1978) assumes infinite apertures, while close inspection of the focal points in Figure 7.47A shows that their geometry varies strongly with position.

The four focal points shown boxed in Figure 7.47A are enlarged in Figure 7.48A. Considering the focal points near the center of the spatial aperture, a small, tight focal point at the top of the section grades to a broad, dispersed smear near the bottom. Alternatively, assessing at constant time shows the focal points grading from strongly asymmetric (left) through symmetric to asymmetric (right). Figure 7.48B shows local *f–k* spectra of the four

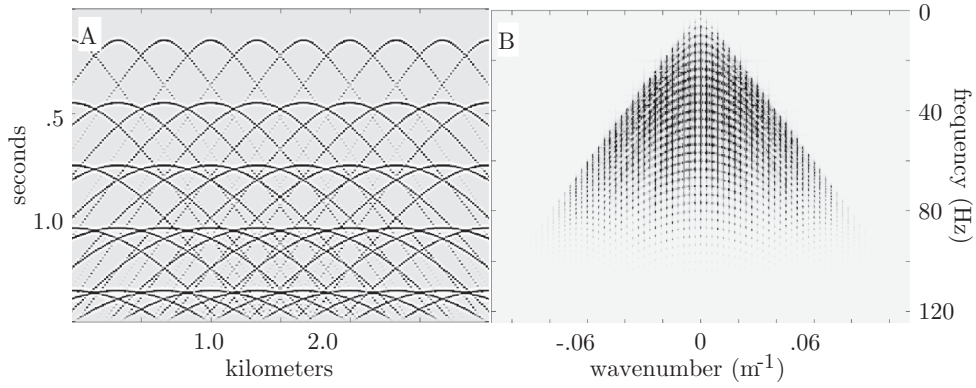


Figure 7.46 (A) An ERM seismogram that models a regular grid of point diffractors. (B) The *f–k* spectrum (amplitude) of panel A.

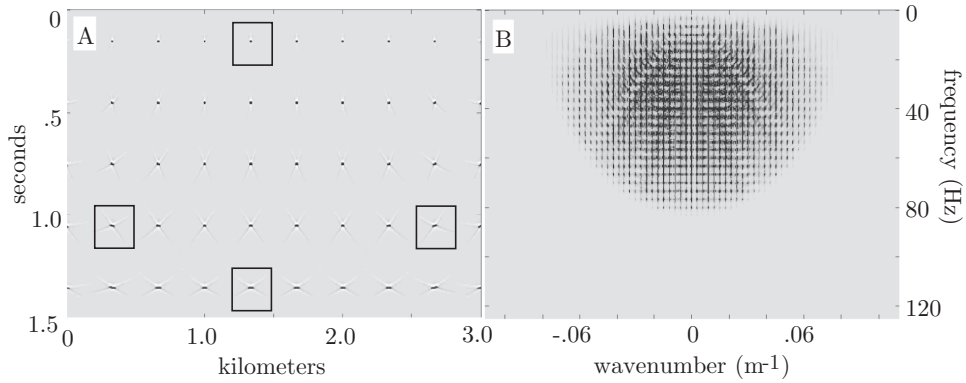


Figure 7.47 (A) The *f*–*k* migration of the ERM seismogram of Figure 7.46A. (B) The *f*–*k* (amplitude) spectrum of panel A.

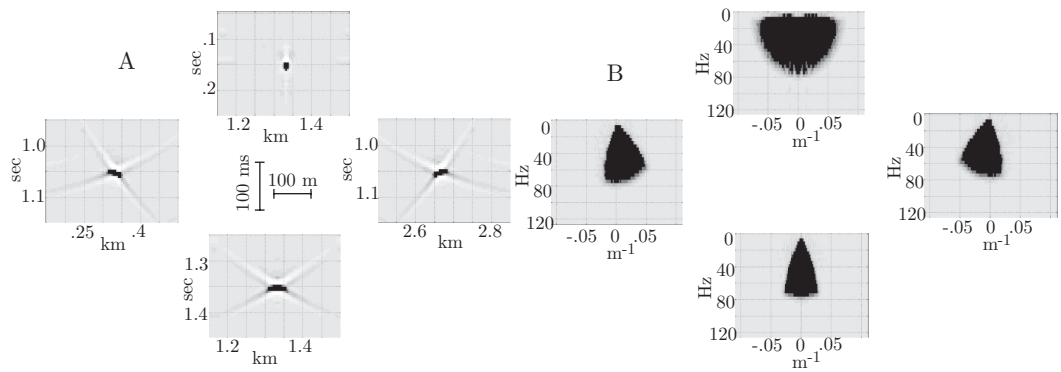


Figure 7.48 (A) Enlargements of the boxed focal points of Figure 7.47A. (B) Local *f*–*k* (amplitude) spectra of the focal points in panel A.

focal points in Figure 7.48A. Comparing with Figure 7.47B shows that these local spectra are dramatically different from the global spectrum. Only the top-center point has the full circular spectrum expected from the infinite-aperture theory, while the others show strong asymmetry or severe bandwidth restrictions. These local spectra determine the local resolution characteristics of the aperture-limited seismic section. Schuster (1997) gives a formal theory (assuming constant velocity) for these focal points and shows that the local spectra are bounded by scattered rays that extend from the scatterpoint to either side of the section. The next section shows how to estimate the corresponding scattering angles in a realistic setting and therefore how to assess resolution implications.

For constant velocity, the computation of limiting scattering angles is well understood, but this approach often results in overly expensive survey designs. An analysis with a constant velocity gradient is much more realistic as it allows for first-order effects of ray bending by refraction. Such an analysis is presented here together with a simple graphical method of assessing the results.

7.7.1 Finite-Dataset Theory

Stolt (1978) established *f–k* migration theory for the poststack, zero-offset case. A fundamental result is that constant-velocity migration is accomplished by a mapping from the (k_x, f) plane to the (k_x, k_z) plane, as was discussed in Section 7.4.1 and illustrated in Figure 7.31. As can be deduced from that figure, the k_x bandwidth after migration is limited by

$$k_{x\,\max} = \frac{2f_{\max}\sin(\theta_{\max})}{v}, \tag{7.133}$$

where $f_{\max}$ is the maximum *signal* frequency and $\theta_{\max}$ is the maximum scattering angle. Both of these quantities are limited by a variety of physical effects, including attenuation (Q), source strength, recording aperture (line length), record length, and spatial sampling. For the present discussion, we will assume a symmetric scattering-angle spectrum, as is expected in the middle of a seismic section. This assumption means that $2k_{x\,\max}$ is an estimate of the k_x bandwidth.

A useful estimate of spatial resolution is given by an expression for the lateral size of a scatterpoint after migration. This should be inversely proportional to $2k_{x\,\max}$ as

$$\delta x = \frac{\alpha}{2k_{x\,\max}}, \tag{7.134}$$

where α is a constant, near unity, whose precise value has little importance.[4] Now, using Eq. (7.133),

$$\delta x = \frac{\alpha v}{4f_{\max}\sin(\theta_{\max})}. \tag{7.135}$$

As expressed here, δx is an estimate of the horizontal size of the smallest resolvable feature on a migrated section. The estimate is made assuming a constant velocity and explicitly gives the dependence of resolution on frequency, scattering angle, and velocity. If the spatial sample size, or CMP bin size, is Δx, then the largest unaliased wavenumber is $k_{x\,\mathrm{Nyq}} = 0.5/\Delta x$ and Eq. (7.135) assumes $k_{x\,\max} < k_{x\,\mathrm{Nyq}}$ or, equivalently, $\Delta x < \delta x/\alpha$. The last expression simply means that the smallest resolvable feature must be larger than the spatial sample size.

For constant velocity, the limits imposed on the zero-offset scattering angle have been given by Lynn and Deregowski (1981). Consider Figure 7.49a, which shows a straight ray from a scatterpoint to a surface receiver grid. For a scattering angle θ, the grid must extend to the right of the scatterpoint by a distance A (e.g., the aperture) given by

$$\tan\theta_A = \frac{A(x)}{z}, \tag{7.136}$$

where the aperture has been denoted by $A(x)$ to emphasize its position dependence in a finite survey. For example, in a 2D survey with a line length L, $A = L/2$ in the center of the line for both left and right scattering. At the beginning of the line, $A = L$ for right

[4] The 2 in the denominator could be absorbed into α but then we lose the expectation that α should be near unity.

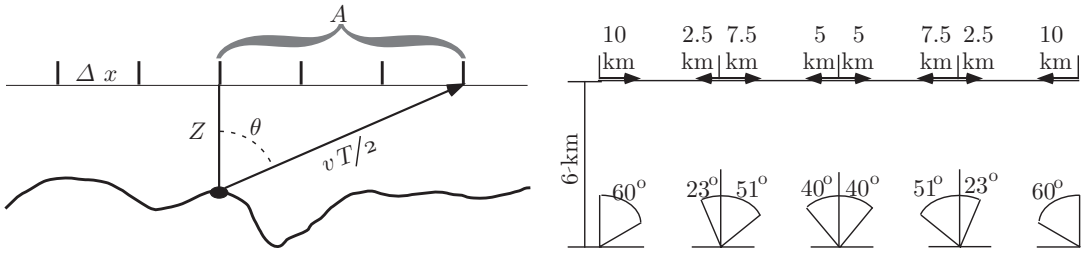


Figure 7.49a (left) A ray from a scatterpoint on a reflector requires a certain aperture *A* and record length *T* to be captured. Also, the CMP sample rate Δx must be sufficiently small to avoid spatial aliasing.

Figure 7.49b (right) A fixed-length seismic line induces an aperture effect that varies laterally along the line. In turn, this limits the scattering-angle spectrum.

scattering and $A = 0$ for left scattering, and just the opposite circumstance occurs at the end of the line. The record length comes into play because the recording system must be turned on for at least the two-way traveltime along the ray. That is,

$$\cos\theta_T = \frac{vT}{2z}. \tag{7.137}$$

In these equations, A is the available aperture, T is the record length, z is the depth, and v is the presumed constant velocity; θ_A and θ_T are the limitations on scattering angle imposed by the aperture and record length, respectively (Figure 7.49a). The aperture is defined as the horizontal distance from an analysis point to the end of the seismic line or the edge of a 3D patch and is thus dependent on azimuth and position (Figure 7.49b). In contrast, the record length limit has no lateral variation. Taken together, these equations limit the scattering-angle spectrum to a recordable subset.

A third limiting factor is spatial aliasing (Section 3.8.1), which further constrains the possible scattering-angle spectrum to that which can be properly imaged (migrated). (Liner and Gobeli (1996) and Liner and Gobeli (1997) give an analysis of spatial aliasing in this context.) The Nyquist requirement is that there must be at least two samples per horizontal wavelength to avoid aliasing:

$$\lambda_x = \frac{\lambda}{\sin\theta_x} \geq 2\,\Delta x. \tag{7.138}$$

Here, Δx is the spatial sample size (CMP interval), λ and λ_x are the wavelength and its apparent horizontal component, and θ_x is most properly interpreted as the emergence angle of a dipping event on a zero-offset section. Consistent with zero-offset migration theory, the exploding reflector model (Lowenthal et al., 1976) can be used to relate wavelength to velocity through $\lambda = v/(2f)$, where f is some frequency of interest. This leads to an angle limited by spatial aliasing given by

$$\sin\theta_x = \frac{v}{4f\Delta x}. \tag{7.139}$$

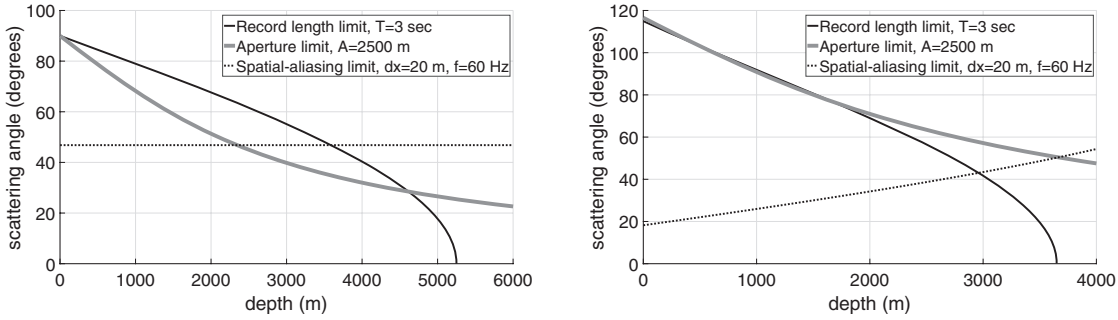


Figure 7.50a (left) Constant-velocity scattering-angle chart showing the aperture, record length, and spatial-aliasing limits for $A = 2500$ m, $T = 3.0$ s, $v = 3500$ m/s, $\Delta x = 20$ m, and $f = 60$ Hz.

Figure 7.50b (right) Constant-gradient scattering-angle chart showing the aperture, record length, and spatial-aliasing limits for $A = 2500$ m, $T = 3.0$ s, $v = 1500 + 0.6z$ m/s, $\Delta x = 20$ m and $f = 60$ Hz.

In the constant-velocity case, the emergence angle of a ray and the scattering angle at depth are equal and thus Eq. (7.139) expresses the constant-velocity limit on scattering angle imposed by spatial aliasing. For vertical velocity variation, i.e., $v(z)$, the result still applies provided that θ_x is simply interpreted as the emergence angle and v as the near-surface velocity. The emergence angle can be related to the scattering angle at depth using Snell's law. This is done by recalling that the ray parameter, $p = \sin(\theta(z))/v(z)$, is conserved (Slotnick, 1959), which leads to

$$\sin\theta_x = \frac{v(z)}{4f\Delta x}. \tag{7.140}$$

This expression generalizes spatial-aliasing considerations to monotonically increasing but otherwise arbitrary $v(z)$, and θ_x is interpreted as the scattering angle at depth z.

Returning to constant velocity, Eqs. (7.136), (7.137), and (7.139) can be used to create a scattering-angle resolution chart for an assumed recording geometry, position on the line, frequency of interest, and constant velocity. A typical case is shown in Figure 7.50a, where it is seen that the aperture limit is concave upward and tends asymptotically to zero at infinite depth. The record length limit has the opposite curvature and reaches zero degrees at a depth $z = vT/2$. Both limits admit the possibility of 90° only for $z = 0$. The spatial-aliasing limit is depth independent but requires a frequency of interest, which can conservatively be taken as the maximum (not dominant) signal frequency. The available angle spectrum is depth dependent and is limited by the smallest angle limit of the three effects at each depth. In this case, spatial-aliasing limits the angle spectrum from 0 to about 2200 m depth, where the aperture becomes the most limiting effect. Aperture dominates to about 4700 m, where the record limit takes over. Near the bottom of any seismic section, the record length limit is always dominant and pushes the dip spectrum to zero at the bottom of the section. From Eq. (7.135), this means there is no lateral resolution at all. The charts for both Figure 7.50a and Figure 7.50b were made by the function `dipspectra`. See also the script `elmigcode/dipspectra_charts`.

Charts such as Figure 7.50a can be used as an aid in survey design but tend to give unrealistic parameter estimates owing to the assumption of straight raypaths. In most exploration settings, velocity increases systematically with depth and thus raypaths bend upward as they propagate from the scatterpoint to the surface. Intuitively, this should lead to shorter aperture requirements and allow the possibility of recording scattering angles beyond 90° in the near surface. The spatial-aliasing limit has already been discussed in this context, and the aperture and record length limits will now be derived exactly for the case of a constant velocity gradient, that is, when $v(z) = v_0 + cz$. The derivation requires solution of the Snell's law raypath integrals (Section 6.11) for the linear-gradient case (Slotnick, 1959). If p_A is the ray parameter required to trace a ray from a scatterpoint to the end of the spatial aperture, then

$$A = \int_0^z \frac{p_A v(z')}{\sqrt{1 - p_A^2 v(z')^2}}\, dz'. \tag{7.141}$$

Similarly, let p_T be the ray parameter for that ray from the scatterpoint to the surface which has a traveltime (two-way) equal to the seismic record length; then

$$T = 2\int_0^z \frac{1}{v(z')\sqrt{1 - p_T^2 v(z')^2}}\, dz'. \tag{7.142}$$

These integrals can be computed exactly, letting $v(z) = v_0 + cz$, to give

$$A = \frac{1}{p_A c}\left[\sqrt{1 - p_A^2 v_0^2} - \sqrt{1 - p_A^2 v(z)^2}\right] \tag{7.143}$$

and

$$T = \frac{2}{c}\ln\left[\frac{v(z)}{v_0}\left\{\frac{1 + \sqrt{1 - p_T^2 v_0^2}}{1 + \sqrt{1 - p_T^2 v(z)^2}}\right\}\right]. \tag{7.144}$$

Equations (7.143) and (7.144) give the spatial aperture, A, and the seismic record length, T, as a function of the ray parameter and velocity structure.

Letting $p_A = \sin(\theta_A(z))/v(z)$ and $p_T = \sin(\theta_T(z))/v(z)$, Eqs. (7.143) and (7.144) can both be solved for the scattering angle to give

$$\sin^2(\theta_A) = \frac{[2Acv_0\gamma]^2}{\left[A^2c^2 + v_0^2\right]^2\gamma^4 + 2v_0^2\gamma^2\left[A^2c^2 - v_0^2\right] + v_0^4} \tag{7.145}$$

and

$$\cos(\theta_T) = \frac{\gamma^{-1} - \cosh(cT/2)}{\sinh(cT/2)}, \tag{7.146}$$

where

$$\gamma = \frac{v_0}{v(z)}. \tag{7.147}$$

When Eqs. (7.140), (7.145), and (7.146) are used to create a scattering-angle resolution chart, the result is typified by Figure 7.50b. The parameters chosen here were the same as for Figure 7.50a, and the linear velocity function was designed such that it reaches 3500 m/s (the value used in Figure 7.50a) in the middle of the depth range of Figure 7.50b. It can be seen that the possibility of recording angles beyond 90° is predicted for the first 1000 m and the aperture limit is everywhere more broad than in Figure 7.50a. The record length limit forces the scattering-angle spectrum to zero at about 3700 m, compared with over 5000 m in the constant-velocity case. This more severe limit is not always the case; in fact, a record length of 6 s will penetrate to over 12 000 m in the linear-velocity case and only 10 500 m in the constant case. Also apparent is the fact that the spatial-aliasing limit predicts quite severe aliasing in the shallow section, though it gives exactly the same result at the depth where $v(z) = 3500$ m/s.

7.7.2 Examples

Figures 7.51A, B, and C provide further comparisons between the linear $v(z)$ theory and the constant-velocity results. In Figure 7.51A, the aperture limits are contrasted for the same

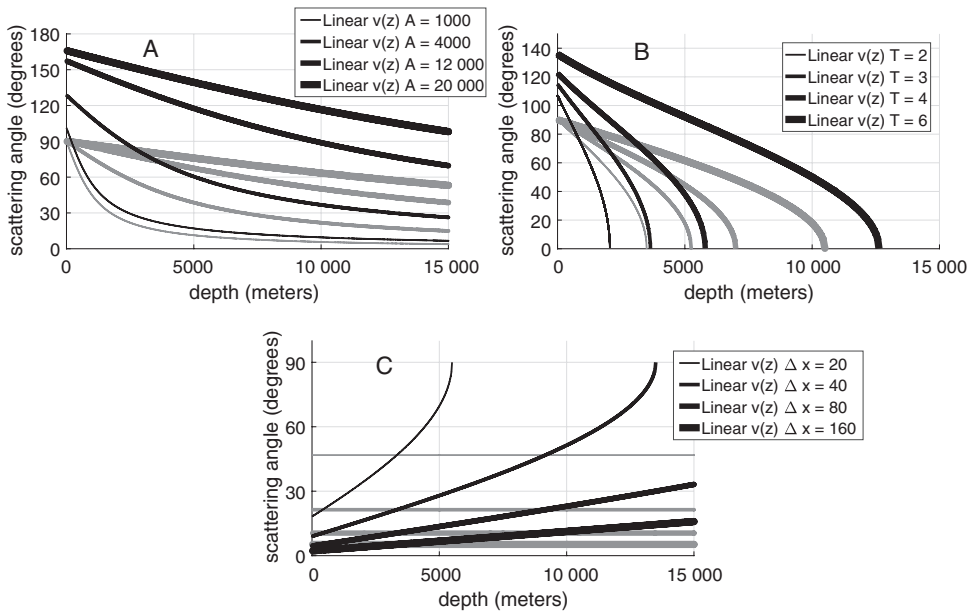


Figure 7.51 The constant-velocity and linear $v(z)$ theories are compared. In each case, the linear $v(z)$ theory is presented with black lines, while the constant-velocity theory uses gray lines. The constant velocity used was 3500 m/s and the linear function was $v = 1500 + 0.6z$ m/s. (A) The aperture limits are contrasted. The apertures used are given for the linear theory but the constant-velocity theory uses the same *A* values in the same sequence. (B) The record length limits are contrasted. The record lengths used are given for the linear theory but the constant-velocity theory uses the same *T* values in the same sequence. (C) The spatial-aliasing limits are contrasted. The Δx values used are given for the linear theory but the constant-velocity theory uses the same values in the same sequence.

linear velocity function ($v = 1500+0.6z$ m/s) and constant velocity ($v = 3500$ m/s) as used before. The dark curves show the linear-velocity results and the light curves emanating from 90° at zero depth are the constant-velocity results. Each set of curves covers the range of aperture values of 1000, 4000, 12 000, and 20 000 m. The dramatic effect of the $v(z)$ theory is especially obvious for larger apertures, which admit angles beyond 90° for a considerable range of depths. Figure 7.51B is similar to Figure 7.51A except that the record length limit is explored. For each set of curves, the record lengths shown are 2.0, 3.0, 4.0, and 6.0 s. Figure 7.51C shows spatial-aliasing limits for a frequency of 60 Hz and a range of Δx values of 20, 40, 80, and 160 m.

Next consider Figure 7.52, which shows a synthetic demonstration of the aperture effect. Here, a number of unaliased point diffractor responses have been arranged at constant time. Thus the record length limit is constant and the aliasing limit does not apply. When migrated, the resulting display clearly shows the effect of finite spatial aperture on resolution. Comparison with Figure 7.49b shows the direct link between recorded scattering-angle spectrum and resolution. Approximately, the focal points appear as dipping reflector segments oriented such that the normal (to the segment) bisects the captured scattering-angle spectrum.

Figure 7.53 illustrates a study designed to isolate the effects of temporal record length on resolution. The unmigrated section is constructed such that all five point diffractors are limited by record length and not by any other effect. Upon migration, the focal points are all symmetric about a vertical axis but show systematic loss of lateral resolution with increasing time. As shown in Figures 7.50a, 7.50b, and 7.51B, the record length limit always forces the scattering-angle spectrum to zero at the bottom of the seismic section.

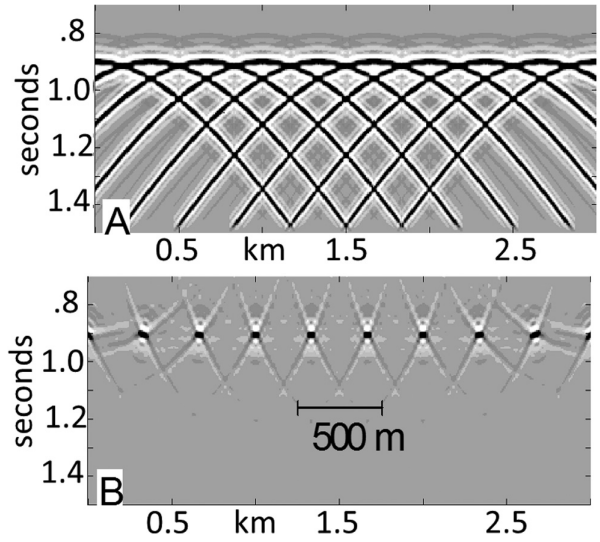


Figure 7.52 (A) A series of point scatterers all at the same time, so that their record length effect is constant. (B) The migration of panel A shows the spatial variability of focal-point geometry caused by the aperture effect. Compare with Figure 7.49b.

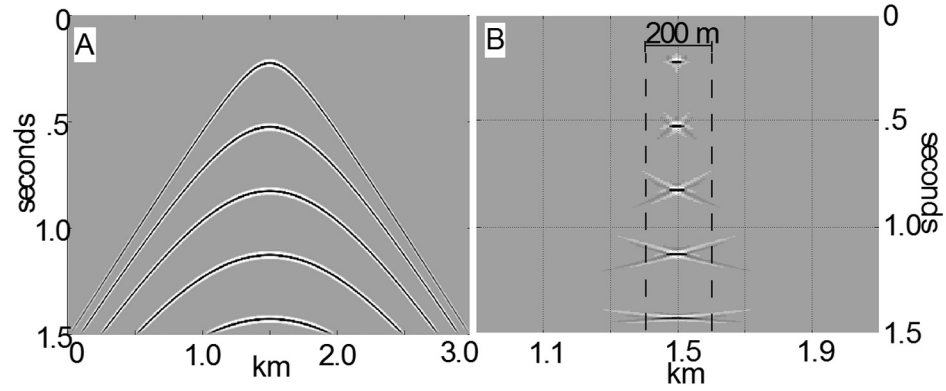


Figure 7.53 An illustration of the record length effect. (A) A series of diffractions that are all truncated by the bottom of the section to minimize the aperture effect. (B) The migration of panel A shows a systematic loss of lateral resolution with increasing time. At the bottom of the section, the lateral size of a focal point is theoretically infinite. Note the change in horizontal scale between panels A and B.

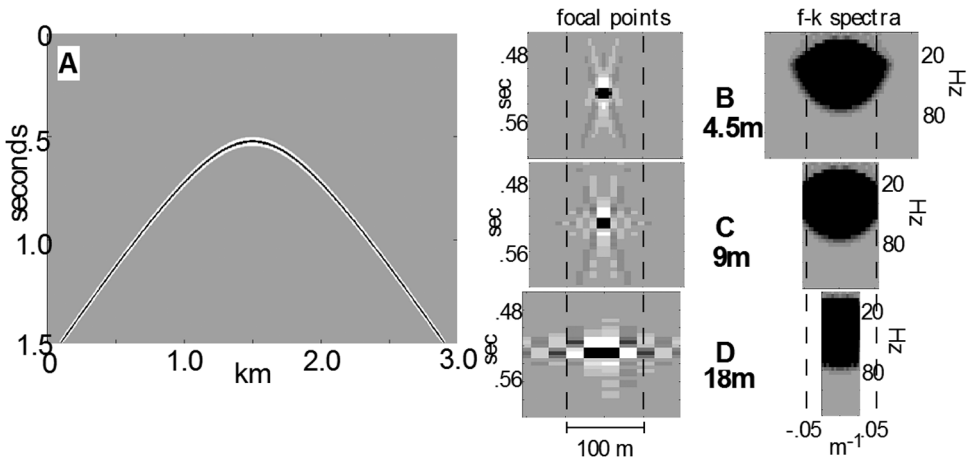


Figure 7.54 The spatial-aliasing effect is demonstrated. (A) A single diffraction hyperbola. (B) A zoom of the focal point and the f–k spectrum after migration of panel A with $\Delta x = 4.5$ m. (C) A zoom of the focal point and the f–k spectrum after migration of panel A with $\Delta x = 9$ m. (D) A zoom of the focal point and the f–k spectrum after migration of panel A with $\Delta x = 18$ m.

Equation (7.135) then results in a lateral resolution size that approaches infinity. This effect accounts for the data "smearing" often seen at the very bottom of migrated seismic sections.

Figure 7.54 shows the migration of a single diffraction hyperbola with three different spatial sample intervals to illustrate the resolution degradation that accompanies spatial aliasing. In part B of this figure, the migration was performed with a spatial sample rate of 4.5 m, which represents a comfortably unaliased situation. In parts C and D, the sample

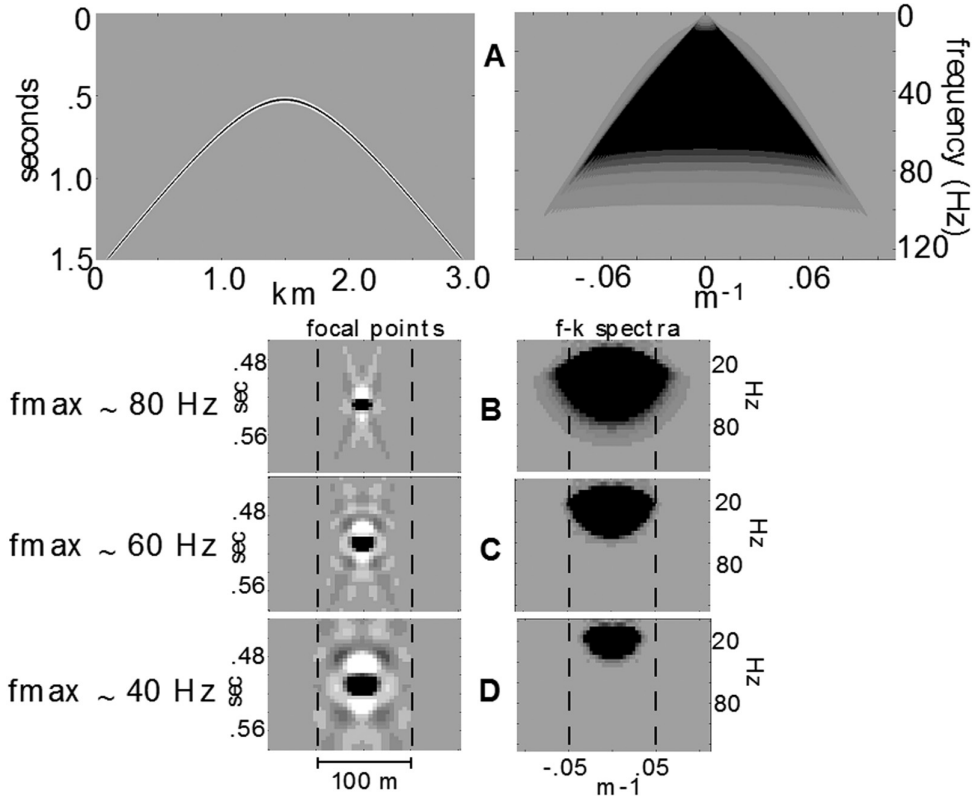


Figure 7.55 The effect on resolution of decreasing f_{max} is demonstrated. (A) A single diffraction hyperbola and its f–k spectrum. (B) A zoom of the focal point and the f–k spectrum after migration of panel A with $f_{max} = 80$ Hz. (C) A zoom of the focal point and the f–k spectrum after migration of panel A with $f_{max} = 60$ Hz. (D) A zoom of the focal point and the f–k spectrum after migration of panel A with $f_{max} = 40$ Hz.

intervals are 9 m (slightly aliased) and 18 m (badly aliased). The slightly aliased situation has not overly compromised resolution, but the badly aliased image is highly degraded. Note that the vertical size of the image is unaffected.

In Figure 7.55, the effect of the maximum temporal frequency is examined. A single diffraction hyperbola was migrated with three different maximum frequency limits. In part B of this figure, the focal point and its local f–k spectrum are shown for an 80 Hz maximum frequency, while parts C and D are similar except that the maximum frequencies are 60 and 40 Hz, respectively. It is clear from these figures that limiting the temporal frequency affects both vertical and lateral resolution. As expected from Eq. (7.135), a reduction of f_{max} from 80 to 40 Hz causes a doubling of the focal-point size.

In summary, the theory of f–k migration predicts a simple model for the resolving power of seismic data. The result is a spatial bandwidth that depends directly on frequency

and the sine of the scattering angle and inversely on velocity. Finite recording parameters (aperture and record length) place space- and time-variant limits on the observable scattering-angle spectrum. Thus the resolution of a seismic line is a function of position within the aperture of the line. The scattering-angle limits imposed by aperture, record length, and spatial sampling can be derived exactly for the case of constant velocity and for a velocity linear with depth. The linear-velocity results are more realistic and lead to considerably different, and usually cheaper, survey parameters than do the constant-velocity formulas.

7.8 Chapter Summary

This chapter has provided an introduction to the basic concepts behind the migration of seismic data. The setting was restricted to stacked data, and so the methods presented are all poststack migration methods. However, each method has a corresponding prestack extension, and codes for these exist in the *NMES Toolbox*. It is hoped that a reader able to grasp this chapter will feel self-empowered to explore these prestack codes. The discussion of migration methods was quite broad, ranging from raytrace migration to wavefront superposition, *f–k* migration, *f–k* extrapolation, Kirchhoff migration, and finite-difference techniques. All of these are simply different techniques to solve the same physics problem and should lead to similar results. However, each has its strengths and weaknesses, and this is the main reason to have so many different approaches.

References

Aki, K., and Richards, P. G. 1980. *Quantitative Seismology*. W.H. Freeman.

Ames, W. F. 1992. *Numerical Methods for Partial Differential Equations*. Academic Press.

Backus, M. M. 1959. Water reverberations: their nature and elimination. *Geophysics*, **24**, 233–261.

Barnes, A. E. 2007. A tutorial on complex seismic trace analysis. *Geophysics*, **72**, W33–W43.

Berkhout, A. J. 1985. *Seismic Migration: Developments in Solid Earth Geophysics*. Vol. 14A. Elsevier.

Borcherdt, R. D. 2009. *Viscoelastic Waves in Layered Media*. First edn. Cambridge University Press.

Bracewell, R. J. 2000. *The Fourier Transform and Its Applications*. Third edn. McGraw-Hill.

Chun, J. H., and Jacewitz, C. A. 1981. Fundamentals of frequency domain migration. *Geophysics*, **46**, 717–733.

Claerbout, J. F. 1976. *Fundamentals of Geophysical Data Processing*. McGraw-Hill.

Clayton, R., and Engquist, B. 1977. Absorbing boundary conditions for acoustic and elastic wave equations. *Bulletin of the Seismic Society of America*, **67**, 1529–1540.

Cohen, L. 1995. *Time-Frequency Analysis*. Prentice Hall.

Cooley, J. W., and Tukey, J. W. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, **19**(90), 297–301.

Dix, C. H. 1955. Seismic velocities from surface measurements. *Geophysics*, **20**, 68–86.

Docherty, P. 1991. A brief comparison of some Kirchhoff integral formulas for migration and inversion. *Geophysics*, **56**, 1164–1169.

Durran, D. R. 1999. *Numerical Methods for Wave Equations in Geophysical Fluid Dynamics*. Springer.

Etter, D. E. 1996. *Introduction to MATLAB for Engineers and Scientists*. Prentice Hall.

Farlow, S. J. 2012. *Partial Differential Equations for Scientists and Engineers*. Reprint edn. Dover.

Futterman, W. I. 1962. Dispersive body waves. *Journal of Geophysical Research*, **73**, 3917–3935.

Ganley, D. C. 1981. A method for calculating synthetic seismograms which include the effects of absorption and dispersion. *Geophysics*, **46**, 1100–1107.

Gazdag, J. 1978. Wave-equation migration with the phase-shift method. *Geophysics*, **43**, 1342–1351.

Goupillaud, P. L. 1961. An approach to inverse filtering of near-surface layer effects from seismic records. *Geophysics*, **26**, 754–760.

Gray, S. H. 1986. Efficient traveltime calculations for Kirchhoff migration. *Geophysics*, **51**, 1685–1688.

Gray, S. H. 1997. True-amplitude seismic migration: A comparison of three approaches. *Geophysics*, **62**, 929–936.

Gustafson, K. E. 1987. *Partial Differential Equations and Hilbert Space Methods*. Wiley.

Hagedoorn, J. G. 1954. A process of seismic reflection interpretation. *Geophysical Prospecting*, **2**, 85–127.

Hanselman, D. C., and Littlefield, B. 1998. *Mastering MATLAB 5: A Comprehensive Tutorial and Reference*. Prentice Hall.

Higham, D. J., and Higham, N. J. 2000. *MATLAB Guide*. Society for Industrial and Applied Mathematics.

Hoffman, K. 1962. *Banach Spaces of Analytic Functions*. Prentice Hall.

Karl, J. H. 1989. *An Introduction to Digital Signal Processing*. Academic Press.

Kjartansson, E. 1979. Constant Q-wave propagation and theory. *Journal of Geophysical Research*, **84**, 4737–4748.

Korner, T. W. 1988. *Fourier Analysis*. Cambridge University Press.

Liner, C. T., and Gobeli, R. 1996. Bin size and linear $v(z)$. *Expanded abstracts of the 66th Annual Meeting of the Society of Exploration Geophysicists*, 47–50.

Liner, C. T., and Gobeli, R. 1997. 3-D seismic survey design and linear $v(z)$. *Expanded abstracts of the 67th Annual Meeting of the Society of Exploration Geophysicists*, 43–46.

Lines, L. R., Slawinski, R., and Bording, R. P. 1999. A recipe for stability of finite-difference wave-equation computations. *Geophysics*, **64**, 967–969.

Lowenthal, D., Lou, L., Robertson, R., and Sherwood, J. W. 1976. The wave equation applied to migration. *Geophysical Prospecting*, **24**, 380–399.

Lynn, H. B., and Deregowski, S. 1981. Dip limitations on migrated sections as a function of line length and recording time. *Geophysics*, **46**, 1392–1397.

Margrave, G. F. 1998. Theory of nonstationary linear filtering in the Fourier domain with application to time-variant filtering. *Geophysics*, **63**(1), 244–259.

Margrave, G. F., and Daley, P. F. 2014. VSP modelling in 1D with Q and buried source. *Research Reports of the Consortium of Research in Elastic Wave Exploration Seismology, www.crewes.org*, **26**.

Margrave, G. F., Lamoureux, M. P., and Henley, D. C. 2011. Gabor deconvolution: Estimating reflectivity by nonstationary deconvolution of seismic data. *Geophysics*, **63**, 15–30.

Mitchell, A. R., and Griffiths, D. F. 1980. *The Finite Difference Method in Partial Differential Equations*. Wiley.

Morse, P. M., and Feshbach, H. 1953. *Methods of Theoretical Physics*. McGraw-Hill.

O'Doherty, R. F., and Anstey, N. A. 1971. Reflections on amplitudes. *Geophysical Prospecting*, **19**, 430–458.

Papoulis, A. 1984. *Signal Analysis*. McGraw-Hill International Edition.

Peacock, K. L., and Treitel, S. 1969. Predictive deconvolution: Theory and practice. *Geophysics*, **34**, 155–169.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press.

Redfern, D., and Campbell, C. 1998. *The MATLAB 5 Handbook*. Springer.

Robinson, E. A. 1954. *Predictive decomposition of time series with applications to seismic exploration*. Ph.D. thesis, MIT.

Robinson, E. A. 1967. Predictive decomposition of time series with applications to seismic exploration. *Geophysics*, **32**, 428–484.

Robinson, E. A. 1983. *Migration of Geophysical Data*. International Human Resources Development Corp.

Robinson, E. A., and Treitel, S. 1967. Principles of digital Wiener filtering. *Geophysical Prospecting*, **32**, 311–333.

Scales, J. A. 1995. *Theory of Seismic Imaging*. Springer.

Schneider, W. S. 1978. Integral formulation for migration in two or three dimensions. *Geophysics*, **43**, 49–76.

Schuster, G. T. 1997. Green's function for migration. *Expanded abstracts of the 67th Annual Meeting of the Society of Exploration Geophysicists*, 1754–1757.

Sheriff, R. E., and Geldart, L. P. 1995. *Exploration Seismology*. Second edn. Cambridge University Press.

Slotnick, M. M. 1959. *Lessons in Seismic Computing*. Society of Exploration Geophysicists.

Stein, E. M. 1993. *Harmonic Analysis: Real-Variable Methods, Orthogonality, and Oscillatory Integrals*. Princeton University Press.

Stolt, R. H. 1978. Migration by Fourier transform. *Geophysics*, **43**, 23–48.

Strang, G. 1986. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press.

Taner, M. T., and Koehler, F. 1969. Velocity spectra – digital computer derivation and applications of velocity functions. *Geophysics*, **34**, 859–881.

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

Taylor, M. E. 1996. *Partial Differential Equations*. Springer. 3 volumes.

Udias, A. 1999. *Principles of Seismology*. Cambridge University Press.

Ulrych, T. J. 1971. Application of homomorphic deconvolution to seismology. *Geophysics*, **36**, 650–660.

Vermeer, G. J. O. 1990. *Seismic Wavefield Sampling*. Society of Exploration Geophysicists.

Vermeer, G. J. O. 2012. *3D Seismic Survey Design*. Geophysical References Series, vol. 12. Society of Exploration Geophysicists.

Waters, K. H. 1981. *Reflection Seismology*. Wiley.

Wiggins, J. W. 1984. Kirchhoff integral extrapolation and migration of nonplanar data. *Geophysics*, **49**, 1239–1248.

Yilmaz, O. 2001. *Seismic Data Analysis*. Society of Exploration Geophysicists.

Zauderer, E. 1989. *Partial Differential Equations of Applied Mathematics*. Wiley.

Index