ENHANCEMENT OF DISCONTINUITIES IN SEISMIC 3-D IMAGES USING A JAVA ESTIMATION LIBRARY

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF GEOPHYSICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Matthias Schwab

June 2001

printed as Stanford Exploration Project No. 99
by permission of the author

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jon F. Claerbout
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Biondo Biondi

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Howard Zebker

Approved for the University Committee on Graduate Studies:

Abstract

Seismic images are difficult to interpret. The nature of the seismic experiment and the ever present sedimentary layering of the subsurface result in oscillatory and complex 3-D seismic images In principle, computer programs can enhance geologically significant image features, such as faults, erosional surfaces, and salt boundaries. Standard edge enhancement, however, fails to compute useful seismic discontinuity maps. The correlation between an image region and its best-fitting local plane wave generates clear and discriminating maps for a wide range of images. A third approach, prediction-error filtering, yields detailed fault maps for images with sharp faults. For images with rather smooth fault zones, prediction error unfortunately fails.

This dissertation's discontinuity computations are the first comprehensive application of a new innovative Java estimation library: *Jest* (Java for estimation). Jest comprises a general and extendible library for numerical optimization for science and engineering, *Jam* (Java and mathematics), and a particular extension of that framework for seismic image processing, *Jag* (Java and geophysics). Jest successfully separates optimization and application software. To ensure the compatibility of solver and application, Jest is built around a set of simple interfaces that define method invocations for the fundamental mathematical objects of numerical optimization, such as vectors, vector spaces, operators, and solvers. Jest's solvers are implemented in terms of these mathematical objects and, consequently, possess the generality of the original abstract mathematical algorithm.

Finally, the dissertation is a reproducible electronic document, a simple software filing

system that organizes, preserves, and potentially transfers the technology of any computational scientific research project. The document's central makefile offers readers four standard commands: `burn` removes the document's result figures, `build` recomputes them, `view` displays the figures, and `clean` removes any intermediate files. of these commands. Although we developed these standards to aid readers we discovered that authors are often the principal beneficiaries. In combination with the World Wide Web's ability to distribute software, standardized reproducibility of computational research offers unprecedented opportunities for collaboration and learning. In particular, the combination of Java software (such as Jest) and the concept of reproducible documents, potentially enables any reader to reproduce a scientist's results at the push of a button in a World Wide Web browser.

Preface

All of the figures in this thesis are marked with one of the three labels: [ER], [CR], and [NR]. These labels define to what degree the figure is reproducible from the data directory, source and parameter files provided on the web version of this thesis [1].

ER denotes Easily Reproducible. The author claims that you can reproduce such a figure from the programs, parameters, and data included in the electronic document. We assume you have a UNIX workstation, the basic SEPlib library, a Java compiler, and the software of the document's web site at your disposal. Before the publication of the electronic document, someone other than the author tests the author's claim by destroying and rebuilding all ER figures.

CR denotes Conditional Reproducibility. The author certifies that the commands are in place to reproduce the figure if certain resources are available. SEP staff have not attempted to verify the author's certification. To find out what the required resources are, you can inspect a corresponding warning file in the document's result directory. For example, you might need a large or proprietary data set, or you might simply need a large amount of time (20 minutes or more) on a workstation.

NR denotes Non-Reproducible. This class of figure is considered non-reproducible. Figures in this class are scans and drawings. Output of interactive processing is labeled NR.

[1] http://sepwww.stanford.edu/public/docs/sep99

In this thesis, the reason to classify a figure reproducible is either that the seismic input image is proprietary or that the computation requires 20 minutes or more on a standard workstation. None of the computations uses proprietary software and all software I use is freely available at my laboratories web site [2]. The reproducibilty of figures requires a Java compiler that correctly implements the *checksource* and *depend* flag. Many current compilers fail to implement these flags correctly.

This dissertation's software is implemented in Java, but it cannot be executed within a World Wide Web browser. The software classes are Java applications and do not satisfy the more stringent security restrictions of Java applets. Furthermore, I currently lack a full-fledged Java graphics package for the display of seismic images. Neither limitation is fundamental and both could be overcome with a bit more time and effort.

[2] http://sepwww.stanford.edu

Acknowledgements

A colleague of mine, Dimitri Bevc, once described his PhD experience as a wonderful, wholesome mountain climb. During my ascent, academic frostbite cost me a toe or two. That I ultimately reached the peak, I owe to a group of exceptional people.

Over the years, Jon Claerbout, my advisor, generously shared his insights and opinions on almost anything under the sun. Unique insights they were! Jon Claerbout should be coauthor of the reproducible research chapter and many other parts of this dissertation. Jon's Stanford Exploration Project (SEP) provides a computational playground beyond a young programmer's wildest dreams. Hospitable SEP seminars are a great forum for frequent and informal exchange among a research team. The SEP sponsor meetings are wonderful opportunities to meet practitioners and to make new friends. All University graduate programs should be run this way!

Joel Schroeder and I implemented the Java optimization library *Jest*, and the GNU make version of the reproducible electronic document system. Comradery and productivity made our collaboration the best months of my PhD candidacy.

Dave Nichols, Steve Cole, and Martin Karrenbach introduced me to SEP. They left much too soon! Steve continues to help me with many problems I encounter, and there is little he cannot fix. Martin taught me SEPlib, reproducible electronic documents, and the production of interactive software CD-ROMs. He and his student, Matthias Jacob, were the first users of Jest, my Java optimization library. Dave is a fountain of ideas. He and Lester Dye are the original designers of CLOP, an object-oriented optimization library and predecessor of Jest.

Bill Symes, Mark Gockenbach, Lester Dye, and I developed HCL, another Jest predecessor implemented in C++. Bill adopted my make-based reproducible electronic document system to organize his laboratory's research and suggested several ways on how to improve it. In 1997, Bill Symes invited me as a visiting scholar to Rice University. Bill listened to many of my ideas and projects with interest, patience, and humor.

Paul Hargrove and I ported the entire SEP working environment to the Linux operating system. Today, Linux by far outnumbers any other operating system at SEP. Paul continues to help whenever I encounter a problem with Linux. Sergey Fomel, Joel Schroeder, and I extended SEP's TEX typesetting system to produce HTML documents. Sergey Fomel superbly administers SEP's TEX system, which I used to typeset this thesis. I wish I had worked more with Sergey. Carey Bunks, Sotiris Kapotas, and Patrick Blondel guided me during an exciting research internship at TOTAL's geophysical laboratory in Paris, France.

Dave Nichols, Steve Cole, Martin Karrenbach, Mihai Popovichi, Dimitri Bevc, Bob Clapp, Sean Crawley, and James Rickett have maintained the SEP computer environment; often beyond the call of duty.

Mary McDevitt edited my SEP reports and versions of this thesis. All remaining errors are probably due to some last minute change of mine. During my tenure, Diane Lau and Linda Farwell, SEP's and the department's administrative assistants, were always extremely helpful and patient.

Davidsons, Douglasses, Claerbouts, and Mouawads gave me a home away from home. I greatly appreciate their hospitality, support, and trust. Confidently, my parents let me march to my own (distant) drum; not an easy thing to do for parents. Many friends have encouraged me during the more stormy days of my ascent, but most my wife, Pascale. I never understood how she could have all that trust in me without bothering with any of the technical details that bogged me down.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis summarizes my work towards three very ambitious goals:

- The computation of discontinuity attributes is a step towards geological computer analysis of seismic image volumes.
- A Java based, object-oriented optimization library may allow unprecedented collaboration of numerical analysts and application experts.
- Web-based reproducible electronic documents may change the very way we conduct and publish computational research.

None of these goals I have completely accomplished. Analysis of discontinuity attributes leads to the confirmation of traditional approaches using plane-wave estimation. Although in certain cases, new algorithms based on prediction error generate discriminating discontinuity maps. The maps differ in their character from the traditional discontinuity maps and facilitate a distinctively different view on the seismic image. Unfortunately, the prediction-error failed to produce clear maps in some test cases. Much research remains to be done. On the other hand, I believe the goals of object-oriented optimization and reproducibility are within our grasp. I think the reader will find my results in these areas useful.

Since the dissertation's three major topics – discontinuity attributes, a Java estimation library (*Jest*) and reproducible documents – are thematically quite independent, each carries its own comprehensive introduction and discussion section. Since I collaborated on each topic with different researchers, I have added individual acknowledgment sections as well.

The three parts nevertheless build an organic unit, since I use the two software tools to implement and organize my research of seismic image discontinuities. In the future, the software will enable any reader to view a set of beautiful seismic discontinuity images on your Web browser. A simple click of your mouse in the caption of any result figure will invoke a Java program and swiftly remove and recompute the figure. Your browser will download the necessary data and Java programs from a laboratory's server and execute routines on your computer. Impressed by the software, you would be free to grab a copy and to apply and extend it to your own research.

The next introductory sections describe the three major topics – the discontinuity attributes, the Java library for estimation, and the reproducible research – in greater detail.

DISCONTINUITY ATTRIBUTES

Maps of discontinuity attributes derived from three-dimensional seismic image volumes aim to delineate fault planes, channel sands, salt boundaries, and boundaries of sedimentary layers. Good discontinuity maps aid in the rapid and reliable interpretation of seismic images.

Interpreters of seismic images derive structural and stratigraphic models of the subsurface geology (Bally, 1983; North, 1985; Vail, 1977; Brown, 1977; Sonneland, 1983). Bahorich and Farmer (1995) introduced an image enhancement technique that delineates discontinuities, such as faults, in seismic image volumes. Subsequently, various other authors (Luo et al., 1996; Gersztenkorn and Marfurt, 1996; Marfurt et al., 1998) published similar tools. All these tools were based on the correlation coefficient among local seismic image patches.

The chapters on seismic discontinuity attributes introduce a series of increasingly sophisticated techniques. The techniques comprise standard edge enhancements (e.g., Laplace operator), misfit of the best-fitting plane wave (correlation), and prediction error (removal of

the linearly predictable image component). I test each technique by applying it to a a simple synthetic image that simulates a normal fault and two genuine seismic image volumes.

Standard edge enhancement techniques fail to detect seismic discontinuities. The oscillations of sedimentary layers constitute amplitude edges that these techniques amplify rather than suppress.

The correlation between a local image patch and its best-fitting plane-wave reliably generates my best discontinuity maps. Other misfit measures of the best-fitting plane-wave proved inferior to the correlation scheme. The method is similar to commercially available seismic discontinuity attributes. There is some uncertainty, however, since publications about these products usually state only the fundamental approach and do not state any technical details.

Prediction error successfully maps sharp image discontinuities. However, smooth fault zones are predicted and, consequently, removed. The discontinuity maps of such smooth images resemble white noise. I design a combination of three two-dimensional prediction-error filters that exclusively removes a plane-wave volume. Contrary to my initial expectation, the filter combination fails to delineate smooth two-dimensional fault zones. Why? By its very nature, a smooth image discontinuity that is a few pixels wide is locally predictable at that local scale and any locally-adapted filter will tend to remove the discontinuity. Furthermore, a prediction-error filter predicts and removes superpositions of plane waves, but not the adjacent-yet-separate plane waves of a fault. In summary, the prediction error of a smooth discontinuity is smaller than I initially expected and the error of a fault's adjacent plane-wave volumes is larger.

The misfit measures between an image and its best-fitting plane wave led me to develop an innovative plane-wave estimation that I believe will be useful in other applications. The approach is related to Symes' (1994) differential semblance optimization and Claerbout's (1994) dip estimation scheme without picking.

The discontinuity approach uses my Java library for model estimation, *Jest*. Particularly, it relies on Jest's patching of local-stationary data. All discontinuity maps are, of course, reproducible.

The computation of discontinuity attributes constitutes the bulk of this thesis and is divided

into four closely related chapters. The problem formulation, standard edge detection, plane-wave misfit, and prediction-error chapter describe the computations of seismic attributes. A fifth appendix-like chapter outlines how I process nonstationary data by breaking them into local-stationary patches.

JAVA OPTIMIZATION LIBRARY

Jest, my Java optimization library, implements abstract mathematical objects, such as vectors, vector spaces, operators, and solvers. The abstraction and encapsulation of those objects into packages enables experts to develop compatible software independently. In contrast to conventional inversion libraries, Jest is not based on a generic data structure.

Traditional solver libraries, such as MINOS (Murthag and Saunders, 1983 revised 1995) or MINPACK (Moré et al., 1980) impose implementation details upon applications. In contrast, Jest simply requires that the invocations of the basic mathematical operations of vectors, operators, and solvers follow certain naming conventions defined by Java interfaces. Our laboratory developed CLOP (1993), an experimental inversion library in C++ and a direct predecessor of Jest. CLOP inspired several similar software packages (Deng et al., 1995; Gockenbach and Symes, 1996; Fomel and Claerbout, 1996). Recently, Jacob and Karrenbach (1997) used Jest to implement a multi-threaded seismic velocity estimation. Jest is implemented in Java (Gosling et al., 1996), a relatively new object-oriented programming language, that is simpler to learn and to program than C++. Java's definition of an abstract deterministic virtual machine enables compilers and interpreters to ensure the safety of Java programs. Consequently, World Wide Web browsers enable Java software to execute on local client machines.

The chapter discusses the innovative and distinctive design of an object-oriented optimization library. Two examples illustrate Jest's capabilities: the deconvolution of a blurred seismogram and the interpolation of missing bathymetry data from neighboring measurements.

Besides Jest's general optimization aspects, its concrete vector and operator objects constitute a budding seismic processing library. All results in this thesis were processed with Jest (though I used SEP's graphics and processing package (*Vplot* and *SEPlib*) to generate

the figures from the numerical result files). However, I believe Jest is still far from its future potential. In particular, Jest ought to include more sophisticated solvers and tackle a few more challenging optimization examples. Since Jest is implemented in Java, future Jest applications ought to convert to reproducible electronic document applets that are accessible by any World Wide Web browser.

REPRODUCIBILITY

Researchers spend valuable time introducing collaborators to the peculiarities of their computer programs. Program documentation is often neglected, since it is not central to the researcher's need to publish. After a few months the original researcher is often unable to reproduce his own computational results. In response, our laboratory, the Stanford Exploration Project, has developed reproducible electronic documents. A small set of standard commands makes a project's results and their reproduction easily accessible to any reader: `make burn` removes the document's result figures, `make build` recomputes them, `make view` displays the figures, and `make clean` removes any intermediate files. The command implementations are usually shared among a research community and an author only adheres to certain file naming conventions during the initial project development stage. The reproducible electronic document does not simply advertise the author's research, but delivers each result as a fully functional example that any reader can inspect and modify.

The first reproducible electronic document was written by Jon Claerbout in 1991. Since then several researchers have improved the original concept and implementation. However, our laboratory's reproducible research concept has unfortunately remained unpublished. The current SEP implementation applies the standard program maintenance tool *GNU make* (Stallman and McGrath, 1991; Oram and Talbott, 1991) to the maintenance of its documents. Two fellow laboratories – Symes' *The Rice Inversion Project* and Donoho's Wavelet research group (Buckheit and Donoho, 1996) at Stanford University – adapted the original SEP version of reproducible documents to their own needs.

Besides a general introductory discussion, the chapter includes a simple illustrative example of a reproducible wave propagation simulation. Finally, the chapter outlines a few details

of the current implementation of SEP's reproducible documents.

To ensure the quality and completeness of our laboratory's biannual research report, a script removes and recomputes all computational results; all result figures in this thesis were tested this way. In principle, a Java implementation of the reproducible document concept would make recomputation of any result of an reproducible electronic document as easy as pushing a button.

Chapter 2

Detection of seismic discontinuities

In the following chapters, I compare several methods to compute discontinuity attribute maps from seismic image volumes. These discontinuity maps aim to delineate boundaries of sedimentary packages, such as fault planes, river channels, and salt flanks. Discontinuities maps accelerate and improve the structural interpretation of traditional seismic image volumes. Figure 2.1 shows a time slice of a Gulf salt dome image and a corresponding discontinuity image – incidently my best result. In this chapter, I introduce the detection of seismic discontinuities by mentioning related publications, discussing the causes of seismic image discontinuities, outlining the investigations of the subsequent chapters, and presenting the three test cases that I use in this study.

Literature

Traditionally, seismic processing back-propagates a surface recorded wavefield to its subsurface reflectors and thereby creates a *wiggly* image of the subsurface. The back-propagated wavefield enables an interpreter to derive a model of the structural and stratigraphic geology of the imaged subsurface (Bally, 1983; North, 1985; Vail, 1977). The interpreter integrates the initial structural image with additional geological and geophysical information to a complete geological subsurface model.

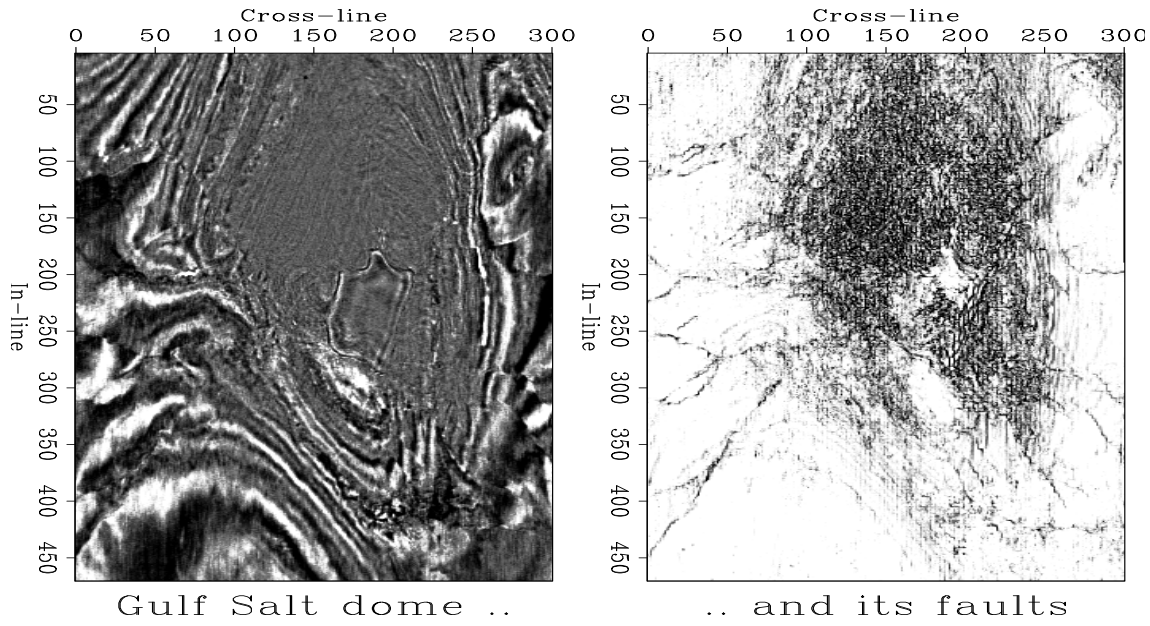

Figure 2.1: Gulf salt dome and its discontinuities. The discontinuity image at the right delineates the faults of the original seismic image at the left. paper-title [CR]

In the last decade, geophysical researchers have attempted to extract information beyond the wiggly, structural image from high-quality seismic recordings. Wavefield inversion (Bleistein, 1984; Tarantola, 1987; Weglein, 1989) estimates physical rock properties, such as local P-, S-wave velocity, and density. Amplitude-versus-Offset analysis relates the prestack seismic data to pore fluid contents or hydrocarbon indicators (Castagna and Backus, 1993). Seismic attribute mapping empirically correlates pre- or poststack seismic data characteristics, e.g., instantaneous frequency, to rock properties (Taner et al., 1979; Sonneland et al., 1990). Many studies combine the various approaches – inversion, AVO, seismic attributes – with other data – e.g., well log data – to extract improved lithological and petrophysical subsurface information (Ronen et al., 1993). The overall goal of this research is to generate continuous, reliable, geological models of the subsurface.

While much work has concentrated on widening the application of seismic data to extract additional information, limited research has been done to improve the extraction of the

traditional structural interpretation. Today's interpreters explore huge three-dimensional image volumes by displaying them on graphics workstations (Brown, 1977; Sonneland, 1983) and by plotting traditional paper sections. These media force interpreters to identify three-dimensional features – river channels, faults – on two-dimensional cross-sections of the image volume. Because of the wavefield character of the seismic image, the interpreter lacks a depth view of the features of interest. To improve picking accuracy, interpreters may use seismic event picking software that evaluates a picked image location by local volume correlation.

Amoco's fault detection technique, *Coherency Cube* (Bahorich and Farmer, 1995), aims to help interpreters by transforming a traditional seismic image volume to a discontinuity image volume. Marfurt et al. (1998) published an improved scheme. In principle, both scheme's resemble the plane-wave misfit, I introduce in this paper. However, I replaced Marfurt's compute-intensive search for the plane-wave dip by a simple analytic estimation. Coherency Cube generated much interest and, subsequently, other researchers (Luo et al., 1996; Gersztenkorn and Marfurt, 1996) presented similar techniques. Bednar (1998) presented a discontinuity attribute based on least-squares dip estimation. Today many commercial seismic processing and interpretation packages include a discontinuity attribute process; their methodologies are usually confidential or patented.

Fundamentally, image enhancement amplifies the pixel amplitudes of the target features relative to the amplitude of the undesired background features. Transform methods, such as Fourier, Wavelet, Hough, or Principal Component decomposition, explicitly separate the image into components that can then be recombined using target enhancing weights. Prediction methods, on the other hand, assume the absence of the target feature and produce high amplitude output for image pixels that deviate from that assumption. However, transform and prediction methods are two modes of thinking rather than two distinct approaches: the Laplace operator, a standard edge enhancement technique (Jain, 1989; Bracewell, 1995), can be thought of as a prediction of the central pixel amplitude derived from its surrounding pixel values. Alternatively, the Laplace operator is often implemented by Fourier transform, in which case it is best thought of as amplifying the high-frequency image components.

Discontinuity, picking, interpolation, and data quality

Discontinuity attribute, seismic reflector picking, interpolation of seismic data, and the assessment of image quality all require a mathematical model of seismic data. A successful model for one application might work for other applications. For example, I derived the discontinuity attribute that is based on the best-fitting plane wave from a dip-picking scheme by Claerbout (1992a). Before using prediction error to detect faults, I (1995) used it to interpret seismic data. Claerbout applies a similar prediction error approach to detect lapses in data quality.

Over the years, I painfully learned, however, that the characterization of seismic data greatly varies among its domains, prestack and image domain as well as two-dimensional and three-dimensional domain. A method that works in one domain does not necessarily succeeds in another. For example, prediction-error filters interpolate the superposed plane waves of prestack data, but fail to remove the adjacent plane waves separated by a fault. Wavelet compression succeeds in the high-dimensional, redundant prestack domain but fails for two-dimensional seismic images.

Future

Interpretation of seismic subsurface images can and ought to be aided by computer processing. The delineation of faults is a step towards general automatic information extraction from seismic images. Automatic seismic image analysis would ease the tedious and time-consuming extraction of geological information from seismic image volumes by human experts. Additionally, automatic analysis could objectively and reproducibly compare the effect of competing seismic image processes on the interpretation of seismic data. However, artificial intelligence experiences great difficulties in the development of visual scene recognition and interpretation (Newell and Simon, 1975).

For continued research into automatic seismic image analysis, I advocate (1) a classification framework for seismic features and (2) a sophisticated segmentation scheme. I wonder if basic statistical measures – mean, mode, variance, autocorrelation, and grey level histograms

– can classify seismic image features and their boundaries – layered sediments, salt, base rock, off-lap surfaces, on-lap surfaces. If such a classification is feasible, the subsurface could be separated into amorphous segments of similar, stationary statistics and, consequently, regions of identical geology. Interpreters of remote sensing data successfully use similar classification and segmentation algorithms (Sabins, 1996).

My investigations

In this chapter I explore three discontinuity attributes: standard edge enhancement, plane-wave misfit, and prediction error.

I illustrate each approach on one synthetic (Figure 2.4) and two seismic images (Figures 2.7 and 2.11). The first seismic image depicts a Gulf salt dome. The faults of the image are sharp and well defined. Bahorich et al illustrates the original coherency attribute at the same image. The second seismic image shows a North Sea horst and is the more challenging test case. The faults are smooth and difficult to find in the seismic time slice. Bednar used the same image in the publication that describes his discontinuity attribute.

The Table 2.1 lists the various methods that I include in this publication. The standard edge enhancement techniques fail to differentiate sedimentary beds and boundaries of sedimentary packages (Figure 3.4). However, two-dimensional edge enhancement within a time slice takes advantage of the dominant horizontal dip of sedimentary strata and yields reasonable fault maps (Figure 3.11).

Of all my discontinuity attributes, the trace correlation of an image region and its locally best-fitting plane wave consistently yields the clearest fault maps (Figures 4.9 and 4.10). Other measures of plane-wave misfit that I tested generated only inferior fault images (e.g., Figure 4.3).

The algorithm of my plane-wave correlation is, similar to the computation of the commercial coherency attribute by Bahorich and Farmer (1995). Both processes certainly generate a similar discontinuity map of the Gulf salt dome. My plane-wave estimation is related to Symes' (1994) differential semblance. I derived the plane-wave estimation by generalizing a

Label	Name	Computation	Output format
Gradient methods			
Grad	Gradient magnitude	$\|\nabla g(x)\|$	scalar at each pixel
Lap	Laplace	$\nabla^2 g(x)$	scalar at each pixel
Dip misfit methods			
DXG	Cross-product	$\mathbf{r} = \mathbf{p} \times \nabla g$	3 component vector at each pixel
DXGLN	Magnitude of r	$\|\mathbf{r}\|$	scalar at each pixel
DXGAA	Backprojection of r	$(\mathbf{p} \times \nabla) \cdot (\mathbf{p} \times \nabla) g$	scalar at each pixel
Waveform misfit methods			
PPC	Correlation of estimate and original	$Cor_z[g(\mathbf{x}), f(\mathbf{p} \cdot \mathbf{x})]$	scalar for each trace of patch
Prediction-error methods			
Pef3d	3-D PEF	Ag	scalar at each pixel
XPef	3 2-D PEF	$\epsilon = [A_x, A_y, A_z]g$	3 component vector at each pixel
XPefLN	Magnitude of ϵ	$\|\epsilon\|$	scalar at each pixel
XPefAA	Backprojection of ϵ	$[A_x, A_y, A_z] \cdot \epsilon$	scalar at each pixel

Table 2.1: Discontinuity attributes. The table lists the various discontinuity attributes that I explore in this thesis. An attribute's label identifies the corresponding names of Figures. The norm $|\cdot|$ is the standard vector norm. The vector $\mathbf{p}$ is an estimate of the local plane-wave normal. The operator A represents a 3-D prediction-error operator. The operators A_i are two-dimensional versions.

dip estimation scheme by Claerbout (1994). Bednar derives his discontinuity attribute from Claerbout's dip estimation, too.

Prediction-error is an unreliable and noisy discontinuity attribute. The discontinuity map of the salt dome (Figure 5.10) shows the radiating faults, and even resolves features within the salt. Unfortunately, the image is obscured by noise. Even worse, the prediction error filter predicts and removes almost all features of the smoother North Sea horst image (Figure 5.11).

Prediction-error images tend to be white (Jain, 1989), since the filter removes the predictable correlated components of its input. Claerbout (1992a) introduced a combination of two two-dimensional filters that detect and suppress a three-dimensional plane-wave. I added a third prediction-error filter, which generalizes Claerbout's intuitive, original formulation. The three prediction-errors are linked to the cross-product expression of my plane-wave formulation.

The prediction-error filters' removal of all image features in the North Sea case suggests limiting the predictive power of the filters. In earlier studies (Schwab et al., 1996a; Schwab, 1997), I limited the number of iterations, and I enlarged and pre-whitened the training patches. However, the prediction-error maps were still white.

The data dependent attribute measures – misfit of local plane wave and the prediction error – depend on local estimation within stationary local image patches. Since patches that contain a discontinuity are nonstationary, the attributes usually yield a large output anywhere within such a patch. (Figures 4.2 or 5.2. Note, that the image is composed of about 300 patches. A side of the cubic patch is 12 pixels long; about one seventh of the total data cube edge of 80 pixels.) The schemes detect if a patch contains a discontinuity, but do not pinpoint the discontinuity within the patch. Consequently, the data dependent schemes are limited to a resolution the size of their patches.

CAUSES OF SEISMIC IMAGE DISCONTINUITY

Sedimentary rocks, the standard environment of hydrocarbon reservoirs, are deposited in horizontal, parallel layers. Subsequent subsidence, uplifting or thrusting may displace or tilt a

sedimentary rock package regionally; however, locally, the sedimentary rocks remain layered. Often sedimentary rocks even remain horizontal, albeit severely dipping beds are not uncommon. The detail of a seismic image volume in Figure 2.2 shows the typical layers of a seismic image of sedimentary rock packages. Figure 2.3 stylizes the layering in a dipping plane-wave volume.

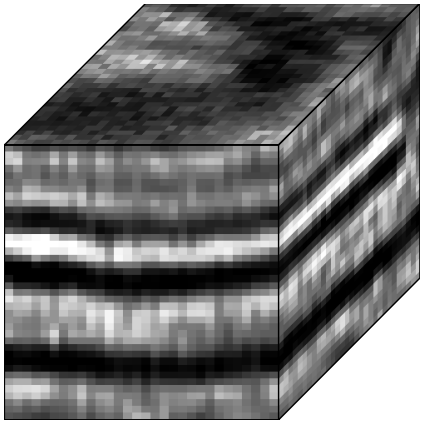

Figure 2.2: Detail of a typical seismic image volume. The image of the sedimentary bedding approximates a nearly horizontal plane-wave volume. paper-nseaFoltZomRaw [ER]

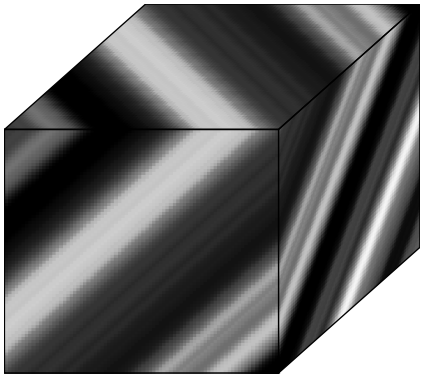

Figure 2.3: Synthetic plane-wave image volume. The plane-wave volume resembles undistorted sedimentary beds. paper-planWaveRaw [ER]

In basins (the standard object of seismic images), faulting, intrusion, and erosion break the original sedimentary layers into smaller sedimentary packages and give rise to plane-wave discontinuities within the seismic image volume. Rocks of a volcanic intrusion, a turbidite flow, or a sedimentary salt give rise to discontinuities by placing their unbedded rocks next to the older sedimentary rocks. Erosion creates subsurface discontinuities by first removing parts of the original sediments and later replacing them with younger sediments. In this case, the discontinuity is due to the difference in the dip and sequence of layers of the old and young

sediments. In particular, river channels wash old sediments away until it replaces them by younger layers of silt. Similarly, a discontinuity due to a fault does not separate fundamentally different rocks. Instead, a fault simply offsets and tilts the previously continuous sedimentary layers. A fault perfectly parallel to its surrounding sedimentary layers, or a fault that perfectly realigns a periodic layer package by a multiple period, would be locally invisible.

In a seismic image volume, a subsurface plane-wave discontinuity usually gives rise to a two-dimensional reflection surface, since in general the discontinuity is a boundary between rocks of different impedance. If the adjacent rocks are similar, as it is often the case with faults, the alignment of the original layers can cause reflections of variable amplitude along the discontinuity. Due to the spatial limits of a seismic survey, reflections from near-vertical discontinuities may not be recorded. Such discontinuities are only visible due to the contrast of the adjacent rock packages. For example, a fault without reflection is visible due to the misalignment of the adjacent sedimentary layer packages.

Additionally, shortcomings in the seismic experiment and processing can cause unwanted image discontinuities. Dead traces, acquisition footprints, unfocused diffractions, and large increments between time slices can result in discontinuities of the seismic image volume that are unrelated to the underlying geology. Image processing can cause otherwise sharp discontinuities to appear smooth. It can either fail to focus the discontinuities or simply smoothen it due to filtering.

In summary, I expect a discontinuity attribute to

- delineate boundaries between neighboring sedimentary packages, such as faults and large channel sands,
- indicate boundaries between sedimentary rocks and rocks without continuous reflections, such as salt and volcanic rocks,
- detect experimental or processing problems, such as acquisition footprints or unfocused diffractions.

Discontinuities vary in their sharpness and contrast due to their origin or the image's processing. In particular, reflectionless faults are sharp discontinuities and faults with reflections are

comparatively smooth discontinuities.

QUALITY OF DISCONTINUITY ATTRIBUTES

The geological interpretation of seismic image volumes depends greatly on the careful mapping of its faults. The exact location and throw of faults are an important ingredient in a geologist's reconstruction of the subsurface's historic sedimentary and structural processes. Additionally, faults potentially trap or leak hydrocarbons. Picking fault surfaces in seismic image volumes is time consuming and error prone, since a fault is easily overlooked. Additionally, the fault lines, which are picked in several two-dimensional image sections, have to be combined into fault surfaces of the image volume.

Current discontinuity attributes cannot replace the picking of faults by interpreters, but they offer an alternative view and they are quick, reproducible, and three-dimensional.

The quality of a discontinuity map ultimately is the ease with which it is interpreted reliably. To that end, a discontinuity attribute needs to suppress the image's sedimentary bedding elements and isolate the image's faults and other discontinuities. Furthermore, the image should resolve any discontinuity as well as possible. In general, the discontinuity image should be free of noise and high in contrast. Beyond such fundamental criteria, every interpreter may have personal preferences.

I found it difficult to develop an image processing algorithm without the guiding principles of physical laws.

THREE TEST CASES

Synthetic image

To illustrate individual discontinuity attribute computations, I will apply them to one synthetic and two seismic image volumes. The synthetic image of Figure 2.4 simulates a simple fault without fault reflection. The image cube shows two distinct regions. One region is filled with

a horizontal plane wave, the other with a dipping plane wave. The contact plane between the two regions models a subsurface fault. An ideal corresponding discontinuity attribute map is zero everywhere but along the fault plane.

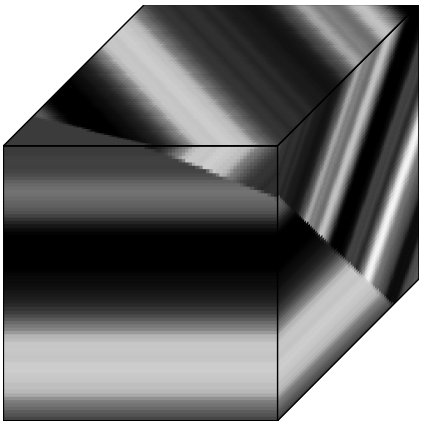

Figure 2.4: Synthetic fault image volume. Two sedimentary layer packages of different dip are separated by a fault without fault reflection. This image serves as an idealized test case for this chapter's discontinuity attributes. paper-zeroFoltRaw [ER]

First seismic image

Figure 2.5 introduces the first seismic test case: A Gulf salt dome and its surrounding sediments and faults. The Figure 2.7 depicts the time slice and vertical sections indicated by the thin mid-volume lines of Figure 2.5.

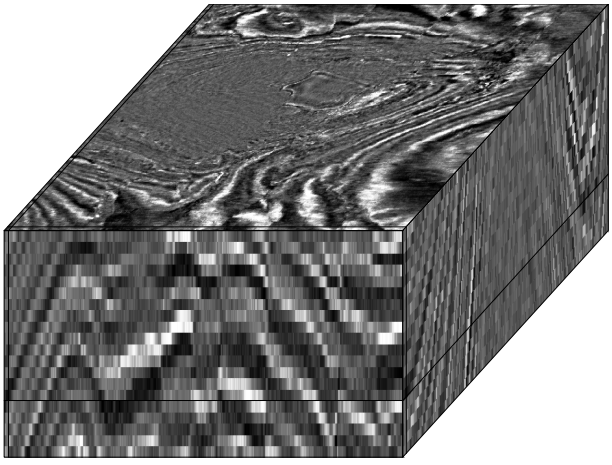

Figure 2.5: Volume view of Gulf salt dome. The Figure represents a volumetric view of the sections of the Figure 2.7. The mid-volume lines indicate the position of the extracted and displayed two-dimensional sections. paper-gulfFoltTotRawVol [ER]

The central salt body is characterized by its low-amplitude internal reflections. The wave-like character of the time slice is due to the outcrops of steeply dipping sedimentary layers.

The vertical east-west section of Figure 2.7 shows how the central salt body has pushed the surrounding sedimentary layers to a near vertical dip. Layers further away from the salt remained approximately horizontal.

The time slice shows a major normal fault that strikes from the north-east to the south-west and that truncates the southern tip of the salt body. Other, smaller faults simply radiate outward from the central salt dome. In addition, the image seems to be somewhat corrupted by weak north-south streaks at the nearly horizontal layers of the time slice's south-east corner. The streaks are probably due to acquisition footprint and constitute image discontinuities as well.

The vertical sections of Figure 2.7 show that the faults dip almost vertically and cut the sediments sharply. The left panel of Figure 2.6 isolates a typical fault of the image. The corresponding averaged trace spectrum on the left indicates that the fault shows considerable energy at high, near-Nyquist frequencies. In this regard, the salt dome's discontinuities resemble the synthetic test case of Figure 2.4.

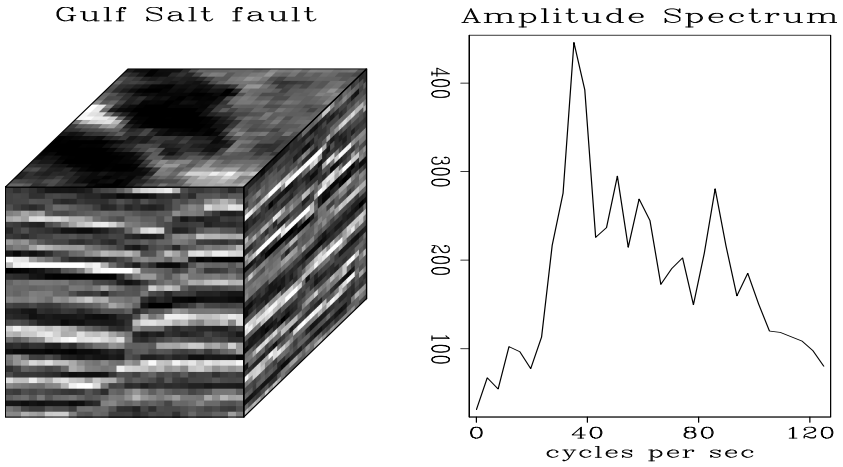

Figure 2.6: Gulf salt dome fault and its trace spectrum. The image detail on the left isolates a typical fault of the Gulf salt dome image. The averaged trace spectrum on the right illustrates the high-frequency contents of the fault image. paper-gulfFoltFltRaw [ER]

Bahorich and Farmer (1995) published discontinuity maps computed from the same seismic image volume. To be competitive with these published maps, a discontinuity attribute has to delineate the salt dome's outward radiating faults and suppress the image's sedimentary

layers independent of their dip. For example, I expect a discontinuity attribute to isolate the forked fault that I marked with an R in the time slice and the in-line section of Figure 2.8. Ideally, a discontinuity map would additionally indicate the fault feature that truncates the salt body at T, and the pentagonoid image feature S.

Second seismic image

The second seismic test case, shown in Figure 2.9 depicts a fault-bounded North Sea horst structure. Figure 2.11 shows a plane view of the the selected time and vertical image slices.

The vertical sections show sets of parallel listric faults that flank a complex horst structure. Except at the northern edge of the image, the sediments are nearly horizontal. The lower fault blocks are, however, slightly tilted, which undulates the upper sediments. In contrast to the vertical section's listric faults, the time slices' intricate zigzag and rhomboidal fault pattern is difficult to discern and indicates the three-dimensional complexity of the entire horst region. The vertical sections of Figure 2.11 show listric faults: their dip decreases with depth. However, the faults do not sharply cut the surrounding sediments but, rather, seem to interpolate between them. As illustrated in Figure 2.10, the images faults show a width of a few samples and their spatial frequency components do not show much energy above half-Nyquist.

I expect a successful discontinuity map to delineate the complex fault pattern of the horst's time slice. Furthermore, I expect the sedimentary layers in the vertical sections to be removed. Bednar (1998) tested his discontinuity attribute at this image. To facilitate a comparison among this chapter's discontinuity attributes, I marked a major fault in Figure 2.11 with an F.

DISCUSSION

Seismic volumes are difficult to interpret, since the ever present sedimentary bedding and a wide variety of other geological features usually crowd the seismic subsurface image. The image is opaque and is usually inspected in extracted two-dimensional sections.

The human eye excels in the recognition of subtle trends in otherwise noisy and incoherent

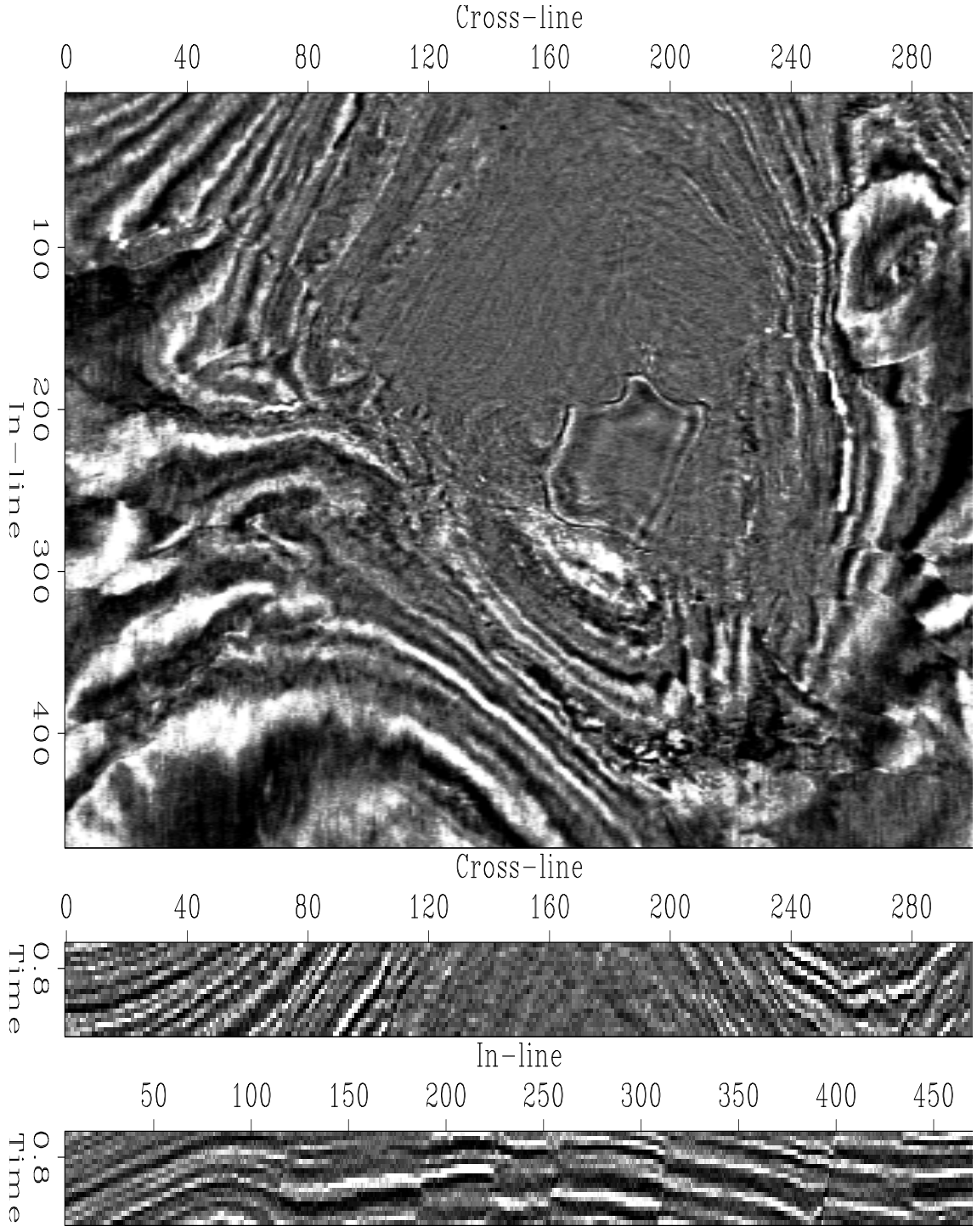

Figure 2.7: Gulf salt dome image. The image depicts a salt body, its surrounding sedimentary layers, and faults. paper-gulfFoltTotRaw [ER]

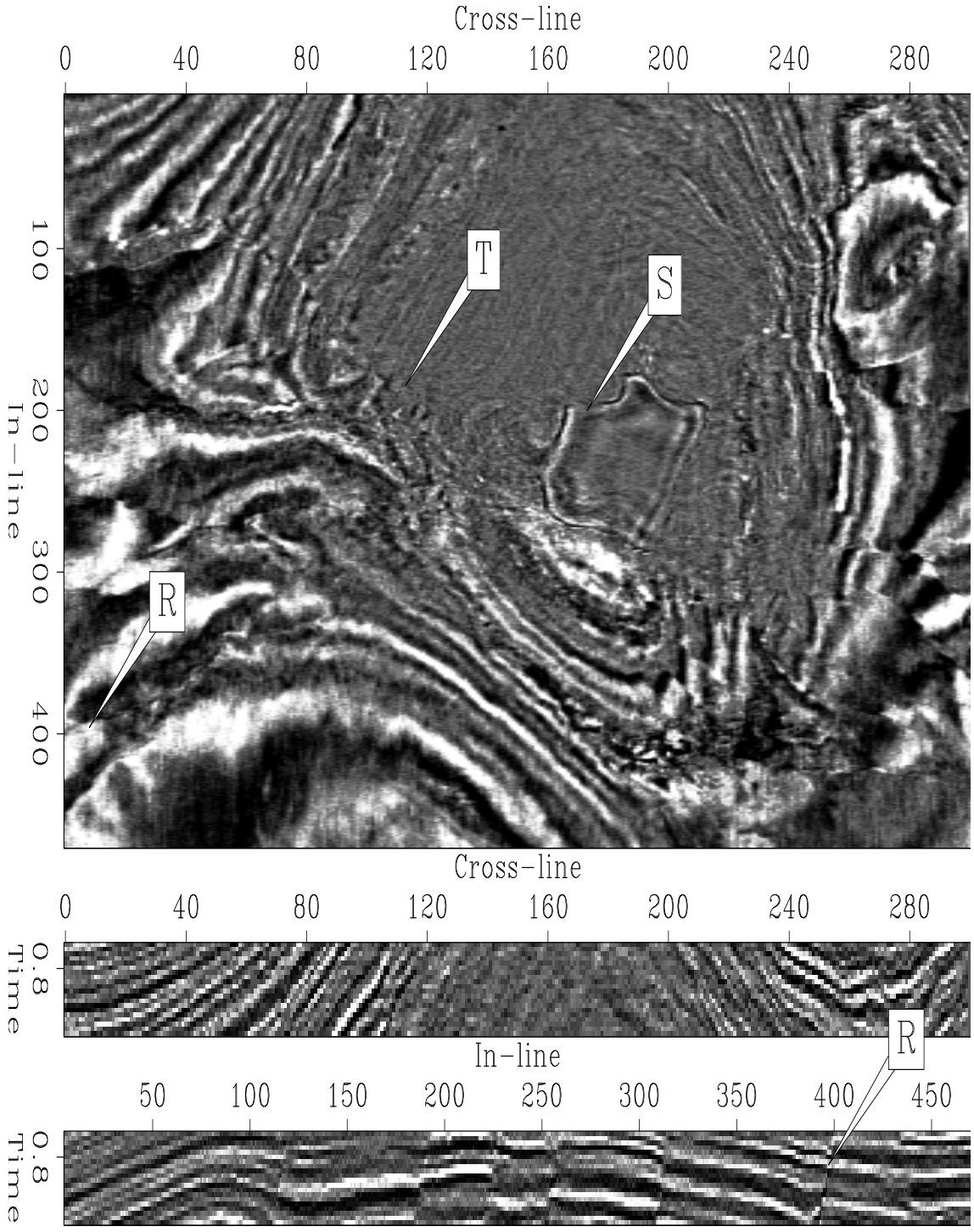

Figure 2.8: Annotated Gulf salt dome image. R marks a radiating fault. T indicates the salt truncation. S points at the salt body's central pentagonoid feature. paper-gulfFoltTotMrk [ER]

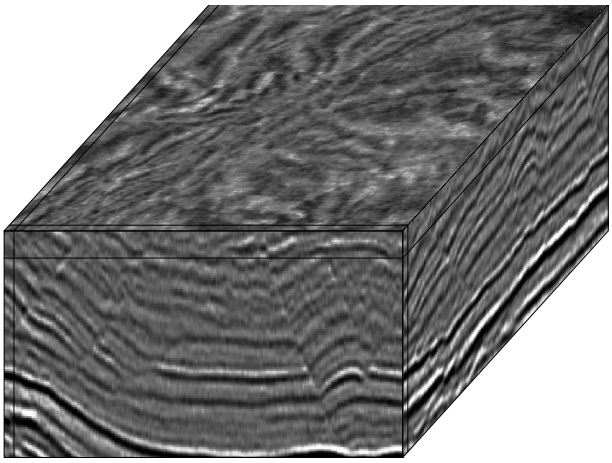

Figure 2.9: Volume view of North Sea horst. The Figure represents a volumetric view of the sections of the previous Figure 2.11. The mid-volume lines indicate the position of the extracted and displayed two-dimensional sections. paper-nseaFoltTotRawVol [ER]

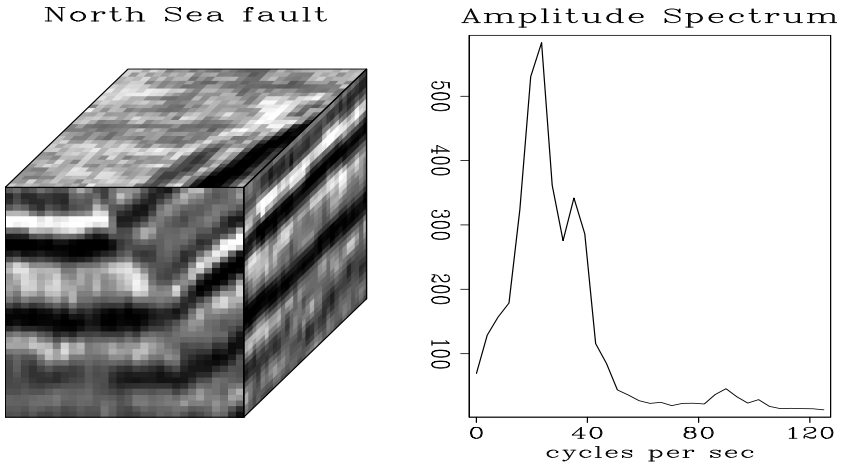

Figure 2.10: North Sea horst fault and its trace spectrum. The image detail on the left isolates a typical listric fault of the North Sea horst image. The averaged trace spectrum on the right demonstrates the rather low-frequency contents of the fault image. paper-nseaFoltFltRaw [ER]

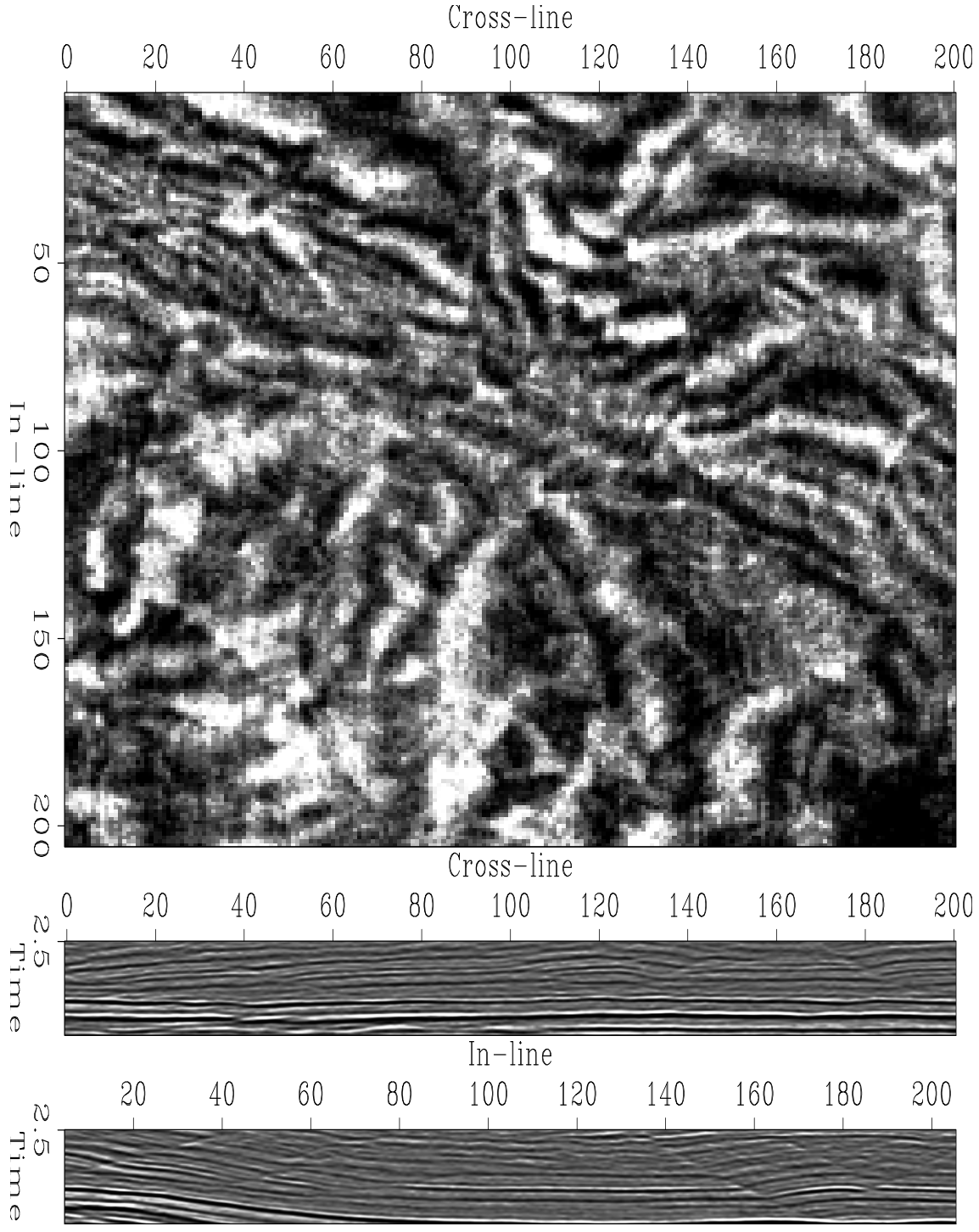

Figure 2.11: North Sea horst image. The time slice depicts a fault-bounded horst structure and indicates an intricate fault pattern. paper-nseaFoltTotRaw [ER]

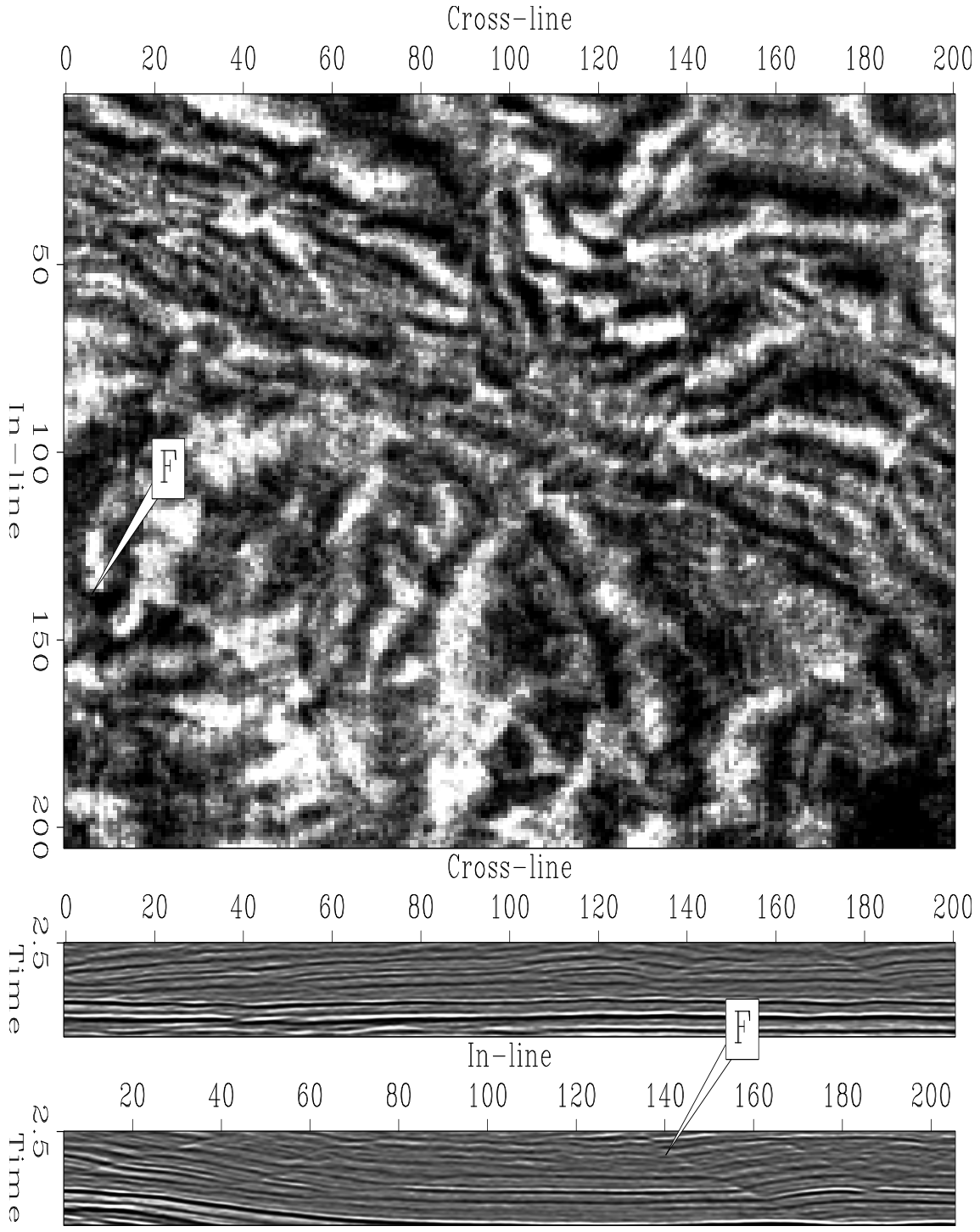

Figure 2.12: Annotated North Sea horst image. F marks a listric normal fault that is barely visible in the time slice, but that is well visible in the in-line section. paper-nseaFoltTotMrk [ER]

data. On the other hand, the eye being restricted to two-dimensional sections is handicapped in grasping the image's volumetric features. For example, within a two-dimensional section, a fault is invisible if it intersects a sedimentary layer in strike direction. If the eye could assimilate the opaque image volume, the presence of the fault would easily be deduced from its existence in nearby image regions. However, the identification of geological features in seismic sections by an interpreter is time consuming and subjective (nonreproducible). Discontinuity maps cannot replace the human interpretation of seismic sections. The maps can, however, help to locate faults, river channels, and other geological boundaries

The synthetic seismic data set represents an idealized sedimentary fault. The fault is simply defined by the offset and of the adjacent plane-waves volumes. In that sense, the synthetic does not represent a fault reflection or transition zone of gnarled rock.

In the two seismic test cases, the major discontinuities are faults. The Gulf salt dome image additionally contains major salt boundaries. With respect to discontinuity attributes, the most important difference between the two seismic examples is the sharpness of their faults. The fault surfaces of the Gulf salt dome image 2.7 are significantly sharper than the fault surfaces of the North Sea horst image 2.11.

The image volumes of the two seismic test cases differ. The size difference of the two image volumes slightly complicates a direct comparison. The larger salt dome time slice appears less grainy than the horst's. Furthermore, little blemishes are bound to disappear in the larger image. Also, the vertical sections of the horst display more than double the time of the corresponding salt sections. Naturally, the images are much more illuminating when displayed on the screen of a graphics workstation.

ACKNOWLEDGMENTS

By its heuristic nature, a study of discontinuity attributes without field data is futile. Thanks to the companies – Haliburton, Geco-Prakla, and Unocal – that supplied migrated image volumes to test the various attribute computations. I especially want to thank Richard Day of Conoco, Kurt Marfurt of Amoco, and Bee Bednar of Bee Consulting for their help in making some of

these image volumes available.

Chapter 3

Standard edge detection

Standard edge detection, such as the gradient magnitude or the Laplace operator, amplifies image regions of large local amplitude change and suppresses regions of constant amplitude. This chapter will demonstrate that standard amplitude edge detection unfortunately does not distinguish between rapid amplitude changes due to layers within a sedimentary package and amplitude changes due to package boundaries, such as faults. Consequently, standard edge enhancement techniques fail to delineate seismic image discontinuities among sedimentary layers.

GRADIENT MAGNITUDE OPERATOR

In Figure 3.1, a gradient magnitude operator detects the amplitude edges at which pixels change their gray-level suddenly. For an image volume $f(\mathbf{x})$, the magnitude of the gradient vector

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2 + \left(\frac{\partial f}{\partial z}\right)^2} \tag{3.1}$$

assumes a local maximum at an amplitude edge. The magnitude is zero, if f is constant.

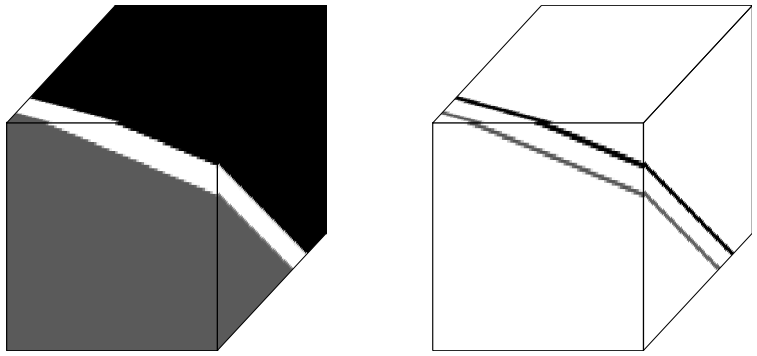

Figure 3.1: Gradient magnitude operator applied to constant amplitude image. The input image on the left consists of three regions of constant amplitude. The output of the gradient magnitude operator on the right delineates the edges: two parallel plane surfaces. The pixel amplitudes are all non-negative. Large amplitudes in the output image indicate edges in the input image. Zero output amplitude indicates locally constant input. paper-consFoltGrad [ER]

In the case of digital images, partial derivatives are conveniently computed by finite-difference approximations. In particular, I use a three-dimensional generalization of

$$\frac{\partial}{\partial x} f(i,j) = f(i,j) * m_x(i,j) \text{ where } m_x = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \tag{3.2}$$

to approximate the horizontal partial derivative of this chapter's image volumes. Table 3.1 lists alternative finite-difference approximations of the partial derivative operator. The various masks vary in their moments, which affects the operators' isotropy, robustness and resolution. Averaging of surrounding values before subtraction reduces the effect of noise and increases the sensitivity to gradual grey-level changes. However, an increase in operator size causes a decrease in the operator's ability to resolve two neighboring edges. The finite-difference approximations illustrate the prediction character of the filter: the difference between the two neighboring pixels is the error one makes predicting one amplitude value to be the same as the next.

Roberts (1965)	Smoothed Prewitt (1970)	Expanded Prewitt	Sobel (Davis, 1975)	Isotropic
$\begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$	$\begin{matrix} -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \end{matrix}$	$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$	$\begin{matrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{matrix}$

Table 3.1: Finite-difference approximations of the horizontal partial derivative.

The Derivative Theorem of the Fourier Transform,

$$\begin{array}{lrcl} \text{If} & \mathcal{F}[\,f(x,y)] & = & F(u,v) \\ \text{then} & \mathcal{F}[\,\frac{\partial}{\partial x} f(x,y)] & = & i2\pi u\, F(u,v), \end{array}$$

shows that the partial derivative operator scales each Fourier component of the input image by its spatial wavenumber in the derivative direction. Consequently, the high frequency components that constitute an edge are amplified and low frequency components that constitute near constant amplitude areas are suppressed. The zero frequency component is erased. Additionally, the derivative operator causes a 90^0 degree phase shift.

Results

The gradient magnitude operator does have its desired edge enhancing effect on non-oscillatory images, such as the synthetic data in Figure 3.1 or the Bay Area map in Figure 3.2. The synthetic test image on the left of Figure 3.1 consists of three constant-amplitude regions that are separated by two parallel planes. The gradient magnitude map on the right delineates the two boundary planes.

The top panel of Figure 3.2 shows the Bay Area topography. The gradient magnitude operator distinguishes topographic features by their slope. In contrast to the Bay Area's smooth sloping hills, the magnitude map's multiple areas of zero gradient surprisingly imply a step-like landscape. I assume that the original topography data was probably sloppily interpolated from a contour map. In general, edge detection and discontinuity attributes often reveal such

image shortcomings.

However, applied to oscillatory images, such as the synthetic test case in Figure 3.3, the gradient magnitude operator indiscriminately amplifies the plane-wave layer boundaries as well as the discontinuity between the plane-wave volumes. Consequently, the plane-wave layers are not suppressed and the central fault is not isolated. The amplitude along the enhanced discontinuity surface varies irregularly depending on the amplitude contrast of the adjacent sedimentary layers.

Similarly, the gradient magnitude operator fails to delineate the faults of the seismic image example of Figure 3.4. The operator increases the overall frequency contents of the image, but does not suppress the sedimentary layer packages. The steeply inclined layers appear aliased in the vertical sections. The faults are hidden among the layer boundaries. Faults, such as the exemplary R fault of the original image 2.8, can be identified only if one knows where to search for them. The so-identified faults are, however, often only one-pixel wide and, consequently, well resolved. The salt truncating fault is not particularly enhanced. Its position is indirectly indicated by a change in the salt bodies average gradient magnitude. Similarly, the boundaries of the salt dome are not clearly pinpointed.

In the case of the Gulf salt dome and the North Sea horst, the gradient operator fails to isolate the faults and yields a map that is as difficult to interpret as the original seismic image.

IMAGE AND PHYSICAL SPACE

In this chapter, I compute discontinuities of a seismic image, not discontinuities of the depicted subsurface. The computations are based on the set of image pixels and do not directly relate to the physical subsurface that the pixels represent. Consequently, partial derivatives indicate change per sample and not change per physical unit, such as meters or seconds. Two differently sampled images of a single physical field $f(x,y,z)$ will result in two different discontinuity images.

The pixel space is defined by the uniform samples of a given discrete image. Its variables $(\hat{x},\hat{y},\hat{z})$ count samples. The vector space has the natural norm $\sqrt{\hat{x}^2+\hat{y}^2+\hat{z}^2}$. Convolutions

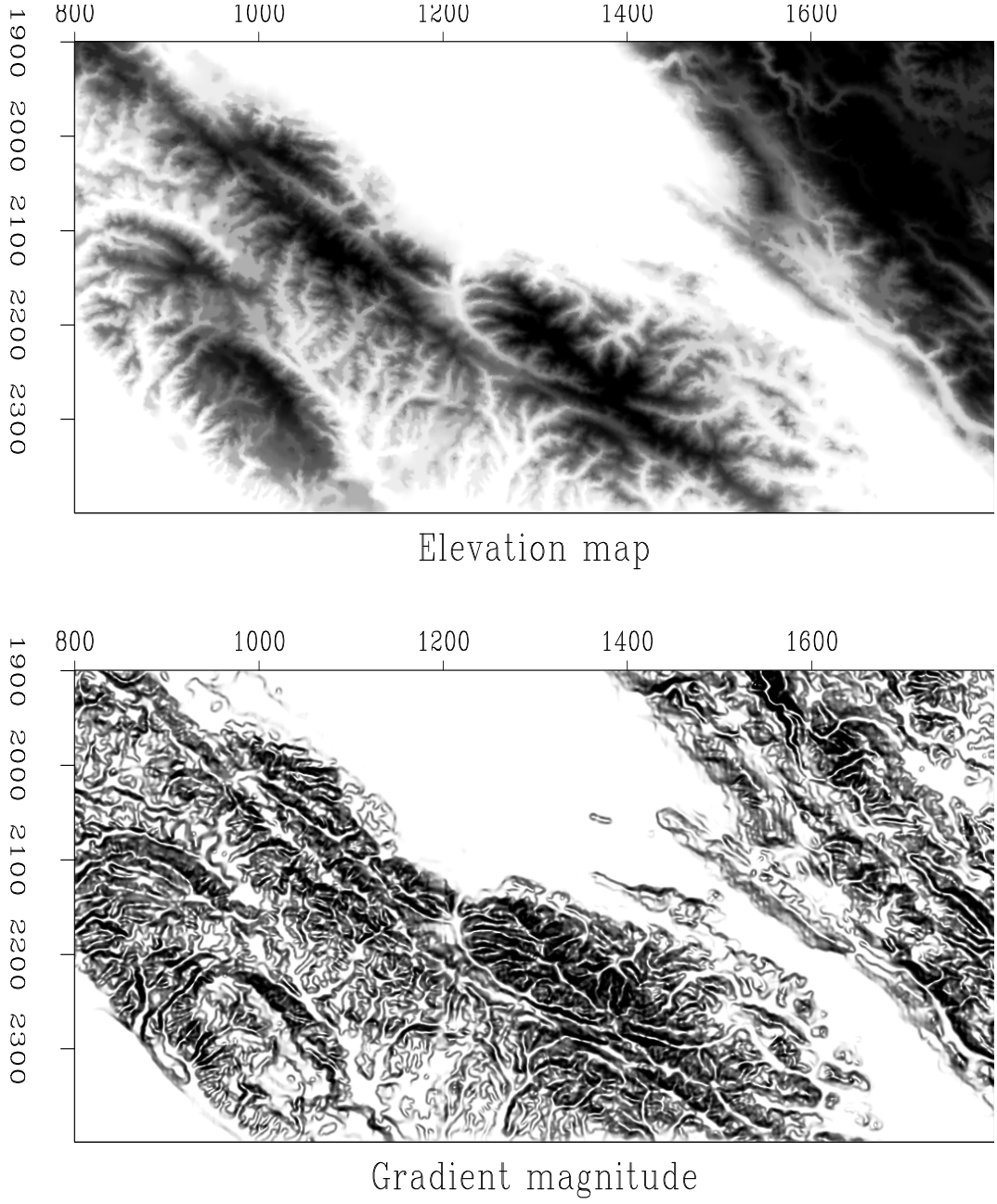

Figure 3.2: Gradient magnitude operator applied to Bay Area topography. The top panel shows the topographic elevation of the San Francisco Bay Area. In the bottom panel, the gradient magnitude operator successfully enhanced the topographic *edges*, such as ravines and canyons. The terraced appearance is an artefact of the original data. paper-bayGrad [ER]

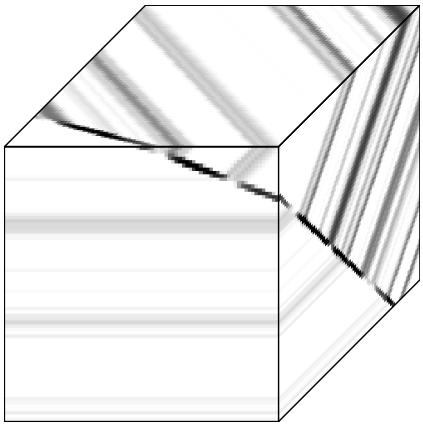

Figure 3.3: Gradient magnitude operator applied to synthetic test case. The gradient magnitude operator fails to isolate the fault, because it cannot differentiate between the amplitude change across the fault and the amplitude change across the sedimentary layers. paper-zeroFoltGrad [ER]

such as expression (3.2) conveniently compute approximations of the partial derivatives of the image. All derivative expressions share the unit *amplitude change over sample*. The gradient magnitude expression (3.1) combines the partial derivatives of the sample space.

A change of variables relates the pixel coordinates $(\hat{x}, \hat{y}, \hat{z})$ to the underlying physical coordinates (x, y, z):

$$\begin{aligned} x &= x_0 + \hat{x}\, dx \\ y &= y_0 + \hat{y}\, dy \\ z &= z_0 + \hat{z}\, dz, \end{aligned} \tag{3.3}$$

where (x_0, y_0, z_0) are the axes offsets and (dx, dy, dz) are the axes increments between samples. The partial z-derivative of the physical space and the pixel space are related by

$$\frac{\partial f}{\partial z} = \frac{\partial \hat{f}}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial z} + \frac{\partial \hat{f}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} + \frac{\partial \hat{f}}{\partial \hat{z}} \frac{\partial \hat{z}}{\partial z} = \frac{1}{dz} \frac{\partial \hat{f}}{\partial \hat{z}}, \tag{3.4}$$

where f is a physical space function and $\hat{f}$ is the corresponding pixel space function.

The physical space is more intractable than the pixel space. For example, the gradient magnitude of a time-migrated image in pixel space is well defined by expression (3.1). The same gradient magnitude cannot be formed in the corresponding physical space, since the units of the partial spatial derivatives and the partial temporal derivative differ.

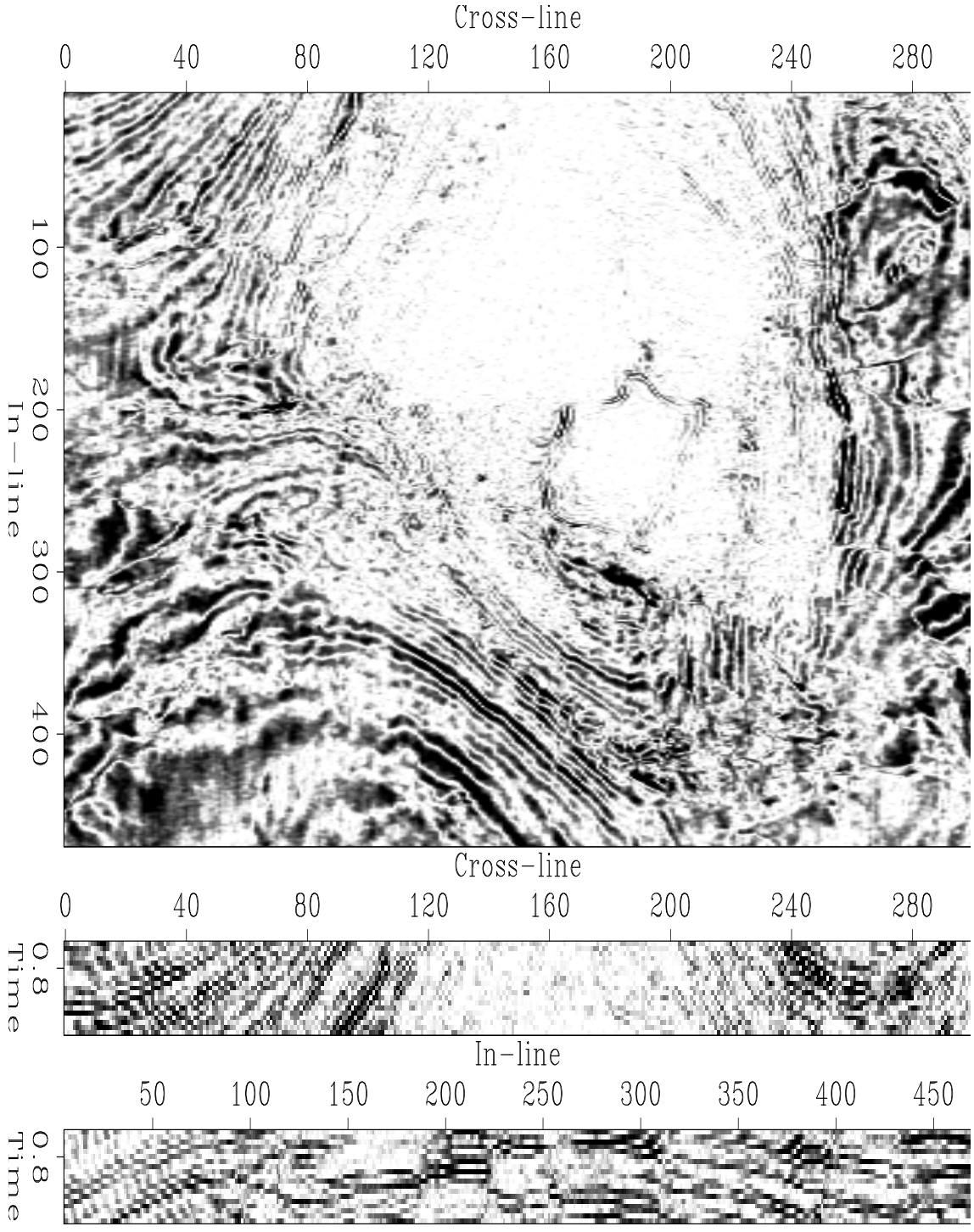

Figure 3.4: Gradient magnitude operator applied to salt image. The gradient magnitude operator fails to suppress the image's sedimentary layers. paper-gulfFoltTotGrad [CR]

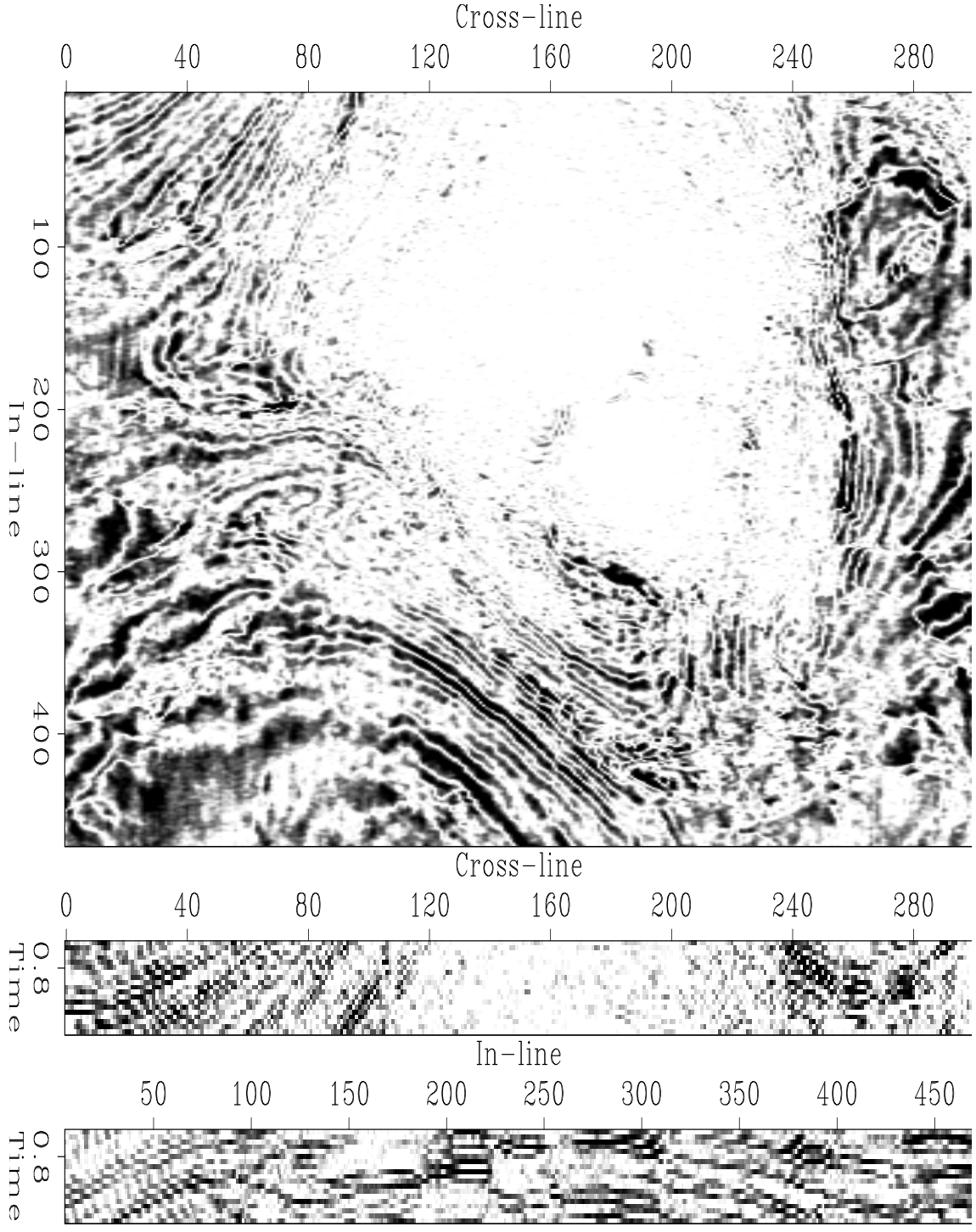

Figure 3.5: Weighted gradient magnitude operator applied to salt image. In contrast to Figure 3.4, this image is the gradient magnitude of the pseudo-depth space, not simply the pixel space. paper-gulfFoltTotWGrd [CR]

To compute the gradient magnitude of the physical space, a time-migrated image would have to be converted from time to depth. A true time-depth conversion requires a subsurface velocity model and the combination of an inverse time migration and a depth migration. Such a costly transformation is unsuitable for a mere discontinuity attribute computation. An alternative pseudo-depth conversion, $z = v/2t$, is a change in variables and leads to a partial derivative weight

$$\frac{\partial f_t}{\partial t} = \frac{v}{2}\frac{\partial f_z}{\partial z}.$$

where f_t is the time-migrated and f_z is the pseudo-depth function. v is a constant approximate subsurface velocity. The expression

$$\frac{\partial \hat{f}_t}{\partial t} = \frac{v\,dt}{2\,dz}\frac{\partial \hat{f}_z}{\partial z}$$

links the partial z-derivative of the time migrated pixel function $\hat{f}_t$ to the partial z-derivative of the pseudo-depth pixel space $\hat{f}_z$.

Seismic images are usually oversampled in time (small dt), and barely sampled adequately in space. Consequently, the time-depth conversion factor dz/dt usually increases the weight of the pixel z-derivative. In the case of the Gulf salt dome image of Figure 2.7, the relation between the two partial derivatives is

$$\frac{\partial \hat{f}_z}{\partial z} \approx 8.3\,\frac{\partial \hat{f}_t}{\partial t}$$

since $dz = 0.05km$, $dt = 0.004s$, $v = 3.0km/s$.

Figure 3.4 shows the gradient magnitude image of the first seismic test case computed in pixel time space. Figure 3.5 shows the corresponding magnitude image based on the pseudo-depth space. Both images show little difference, except that the salt body appears noisier in the unweighted image.

In general, I choose the *unphysical* pixel space to compute this chapter's discontinuity attributes. Why? The computation of a discontinuity attribute is not based on a physical process and it makes little sense to talk about its correctness. Its usefulness results purely from its interpretational clarity and its computational simplicity.

In my experience, a large weight of the temporal pixel derivative decreases the quality of the discontinuity map. I believe the deterioration is due to the importance of the horizontal derivative terms in annihilating the prevalent horizontal sedimentary layers. Empirically, the rather successful discontinuity attribute of Figure 3.11 demonstrates the importance of horizontal comparison. The attribute is horizontal correlation.

Furthermore, the pixel space measurements are conceptually and computationally simple. Traditionally, enhancement techniques in the image processing literature use the same pixel space approach (Castleman, 1996).

PROCESSING AND DISPLAY PARAMETERS

To ensure that the result images, such as Figure 2.7 and 3.4, are comparable, I process identical image volumes for all computations. All results are represented by the same image sections of the image volume.

Usually, each processing step offers its user a set of configurable parameters. In the case of the gradient magnitude operator, users can set individual weights for each partial derivative term or choose among competing finite-difference approximations of the partial derivatives. In general, I experiment with such processing parameters. Because of the shear amount of possible combinations, such experiments are usually not exhaustive. The result images that I include in this report are the *best* results that I found. Similarly, I choose individual display parameters, such as polarity and clip, for each image.

LAPLACE OPERATOR

In Figure 3.6, a Laplace operator transforms the amplitude edges in the image cube on the left to zero crossings of a bipolar wavelet in the image cube on the right. Given an image volume $f(\mathbf{x})$, the Laplace operator

$$\nabla^2 f(\mathbf{x}) = \frac{\partial^2 f(\mathbf{x})}{\partial x^2} + \frac{\partial^2 f(\mathbf{x})}{\partial y^2} + \frac{\partial^2 f(\mathbf{x})}{\partial z^2}$$

									−1	−1	−1	−1	−1
0	−1	0	−1	−1	−1	1	−2	1	−1	−1	−1	−1	−1
−1	4	−1	−1	8	−1	−2	4	−2	−1	−1	24	−1	−1
0	−1	0	−1	−1	−1	1	−2	1	−1	−1	−1	−1	−1
									−1	−1	−1	−1	−1

Table 3.2: Finite-difference approximations of the 2-D Laplace operator.

zeroes the amplitude where f is constant. Analogous to the gradient magnitude operator case, a Laplace operator is conveniently approximated by its finite difference approximations, such as those shown in Table 3.2. In general, the masks vary in their moments and consequently in their isotropy, robustness and resolving power. In this chapter, I use a three-dimensional version of the one-dimensional and two-dimensional Laplace filters of Figure 3.7.

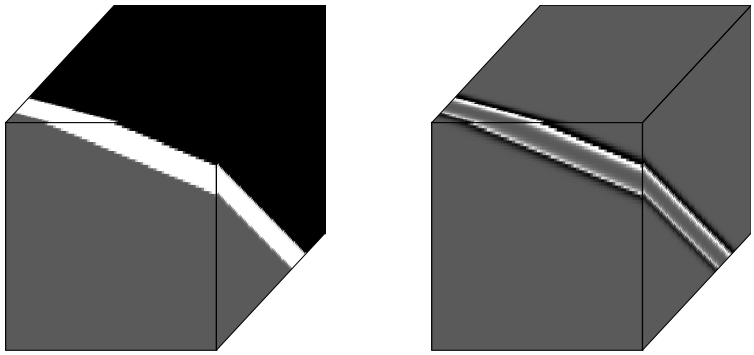

Figure 3.6: Laplace operator applied to constant amplitude image. The output of the Laplace operator on the right delineates the edges successfully. Edges are indicated by the zero-crossing (grey) of a bipolar wavelet. Constant input regions result in zero output values (grey). paper-consFoltLap [ER]

The finite-difference representations of Table 3.2 approximate a generalized Laplace operator that combines the nabla operator with a pre-smoothing by a Gaussian:

$$g(x,y) = -\nabla^2 \left[\frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \right] = \frac{1}{\pi\sigma^4} \left[1 - \frac{x^2+y^2}{2\sigma^2} \right] e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{3.5}$$

where σ controls the width of the Gaussian kernel. In the Fourier domain, the generalized

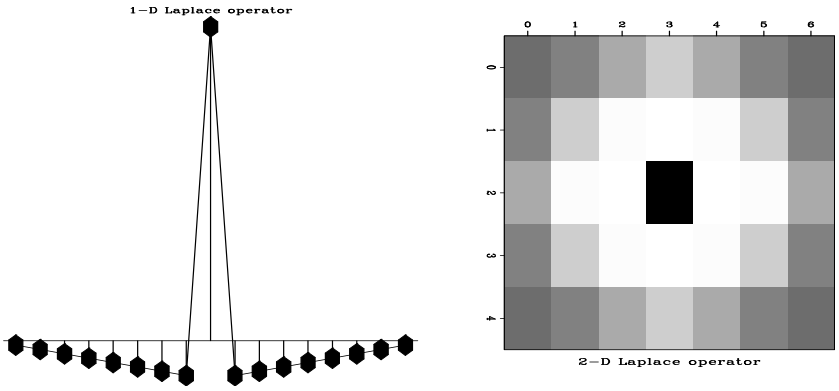

Figure 3.7: Generalized Laplace operator. The one-dimensional filter on the left is constructed by subtracting a wide and a narrow triangle of equal area. Similarly, the two-dimensional filter on the right is constructed by subtracting a wide and a narrow pyramid of equal volume. paper-lapGeneral [ER]

Laplace operator (3.5) is

$$\mathcal{F}[g(x,y)] \propto (u^2+v^2)e^{-2\sigma^2(u^2+v^2)}.$$

The factor (u^2+v^2) is the Fourier transform of the nabla operator; the exponential is the transform of the Gaussian low-pass filter.

Results

The Laplace operator detects edges between constant amplitude volumes (Figure 3.6). Applied to the synthetic test case, the Laplace operator delineates the discontinuity but only partially suppresses the plane waves (Figure 3.8). The wavelet along the discontinuity changes its polarity depending on the contrast of the adjacent plane waves. The somewhat lower amplitude of the Laplace operator among the horizontal plane wave is probably due to the exact zero of the horizontal component.

Applied to the first seismic test case (Figure 3.9), the Laplace operator creates a discontinuity map that generally resembles the original seismic image. The wavefield-like character of the time slice remains. The fault marked R in the original image 2.8 is not enhanced: neither

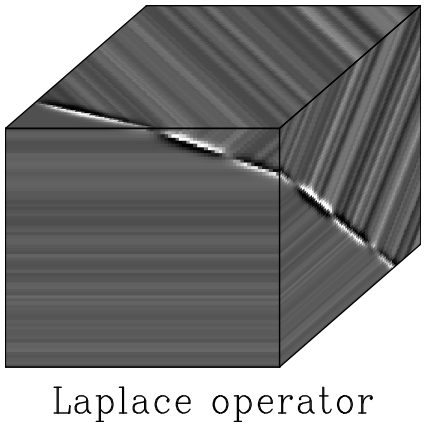

Figure 3.8: Laplace operator applied to synthetic test case. The Laplace operator slightly enhances the fault plane between the neighboring plane-wave volumes. However, the plane-wave oscillations lead to noticeable residuals in the plane-wave volumes. paper-zeroFoltLap [ER]

are any other faults. Applied to the image of the North Sea horst, the Laplace operator yields a similarly disappointing discontinuity map (not shown).

HORIZONTAL CORRELATION

In the special case, that all sedimentary layers are horizontal, a time slice of an undisturbed horizontal layer shows a constant amplitude, and, within the time slice, discontinuities create standard amplitude edges. The amplitude edges could be enhanced by standard edge enhancement techniques applied to individual time slices. Instead, I chose trace correlation to compute discontinuity attributes of horizontally layered seismic images.

Correlation measures horizontal alignment of one-dimensional arrays (traces). The standard normalized correlation between two one-dimensional arrays, f_1 and f_2, is

$$c = \frac{\sum_i f_{1i} f_{2i}}{\sqrt{\sum_i f_{1i} \sum_i f_{2i}}} \tag{3.6}$$

where i is the arrays' sample index. Neidell and Taner (1971) generalize the correlation coefficient to measure alignment among a set of one-dimensional arrays:

$$c = \frac{2}{N(N-1)} \sum_i \sum_{j>i} \left[\frac{\sum_k f_{ik} f_{jk}}{\sqrt{\sum_k f_{ik}^2 \sum_k f_{jk}^2}} \right], \tag{3.7}$$

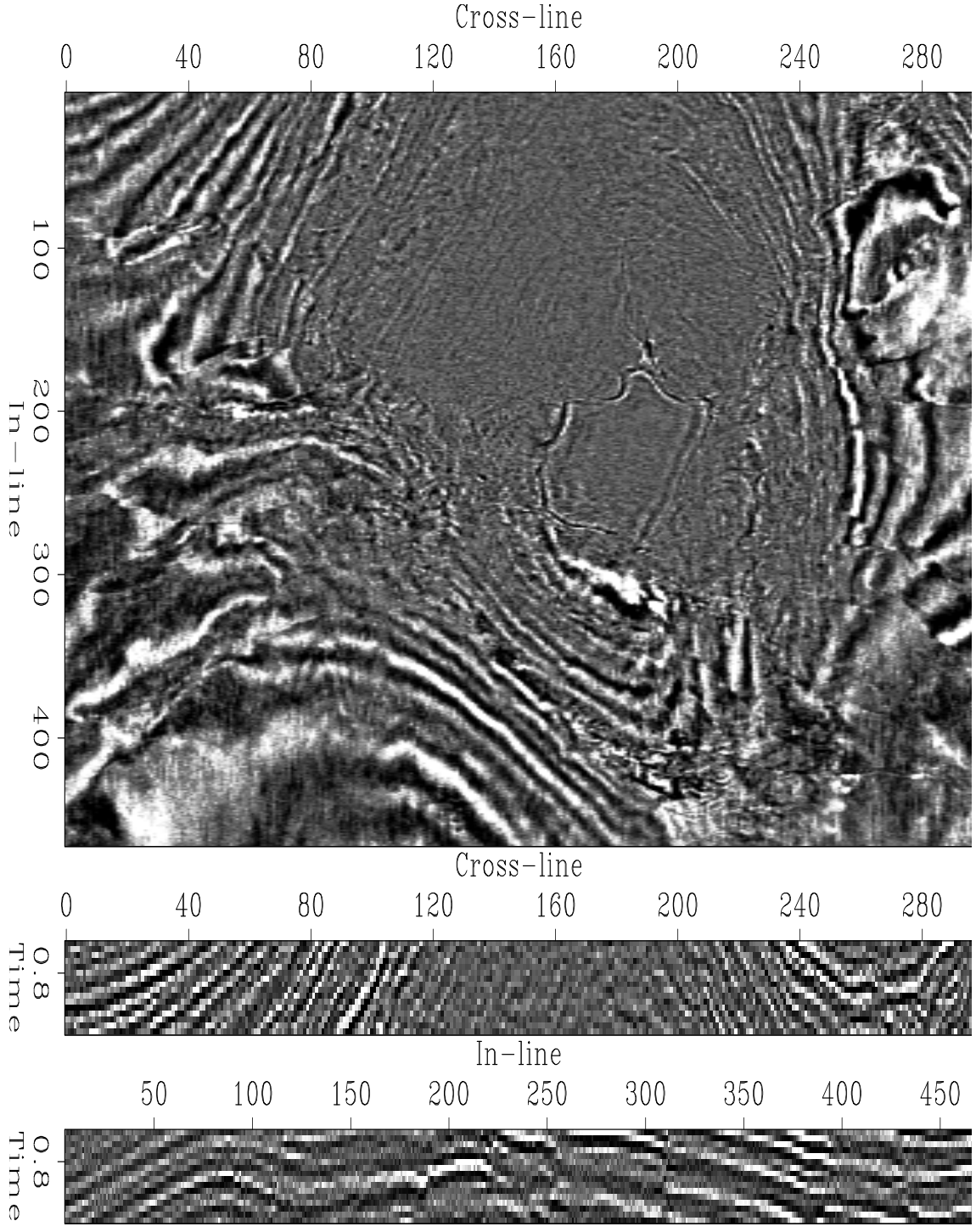

Figure 3.9: Laplace operator applied to salt image. The Laplace operator does not change the character of the original seismic image. In particular, the discontinuity map fails to delineate the sought faults. paper-gulfFoltTotLap [ER]

where N is the length of a single trace in samples and f_{ik} is the k-th sample of the i-th trace. The correlation coefficients are bounded by -1 and 1. The coefficient c is 1 only if all the traces are identical.

To simplify plots of correlation coefficients, I map the coefficient range from $[-1,1]$ to $[0,2]$ so that pixels of correlation 1 – perfect correlation – plot as 0 and less well-correlated regions show positive amplitudes. Hence, I first subtract 1 from each pixel value and then multiply it by -1.

To compute the local correlation of a nonstationary image volume, I split the image into small patches, compute the generalized correlation within the patch, set all patch pixels to its correlation value c and merge the individual patches to a single output quilt[1]. The correlation within a patch measures the similarity among its traces f_i, the one-dimensional vertical array of all pixels of identical horizontal location. Perfectly horizontal layers result in a correlation coefficient of 1 and are mapped to a pixel value of 0.

The patch size is determined by a trade-off between resolution and data nonstationarity on one side and the need to capture sufficient statistical information in a patch on the other. Since each patch is filled with a single correlation coefficient, the procedure can only resolve two discontinuities if they are separated by more than the size of a patch. Besides loss of resolution, a large patch may capture nonstationary data and yield incorrect statistical estimates. On the other hand, a small patch may not contain enough data to gather reliable statistics in the presence of noise.

Results

The horizontal correlation attribute suppresses the horizontal layers, but fails to remove dipping ones (Figure 3.10). The discontinuity is delineated where the adjacent layers sufficiently contrast. The large and blurred character of the image's features illustrates the reduced resolution due to the patch size.

[1]This dissertation's chapter on nonstationary data processing discusses the details of the patching procedure.

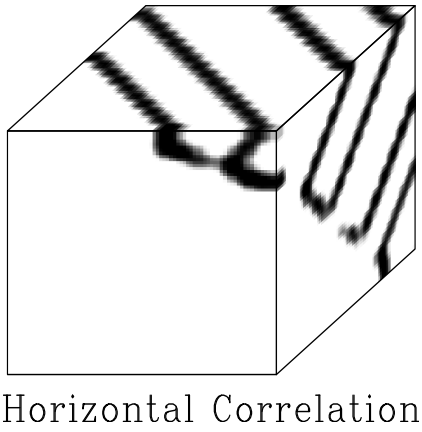

Figure 3.10: Horizontal correlator applied to synthetic test case. The correlation of neighboring trace intervals annihilates the horizontal beds but fails to reject the dipping ones. Hence, the fault is not well isolated. paper-zeroFoltCor [ER]

Horizontal correlation is a surprisingly successful discontinuity attribute when applied to seismic subsurface images. Applied to the Gulf salt dome image (Figure 3.11), the correlation successfully suppresses the horizontal layers. Even dipping layers are removed, if laterally they smoothly vary, due to their width. However, high-frequency dipping layers, such as in the north-west corner, are not removed, but show a considerable discontinuity amplitude.

The radiating faults are discerned. The fault marked R in the original seismic image 2.8 is depicted. An intrepid interpreter may even suspect that the fault diagonally crosses the salt body and reappears in the north-east corner of the time slice. The primary faults in the subsurface are accompanied by similar linear features with lower amplitude. These events might be fault surfaces that only partially intersect the local patch or that separate sedimentary layers of little local contrast. Overall, the faults contrast sharply with their surrounding image regions. The discontinuity map has about patch-sized resolution, as expected.

The salt itself gives rise to a region of little correlation and consequently high amplitude. Among the chaos of events, the salt truncating faults are not enhanced. The central pentagonoid region at the southern tip of the salt body mostly vanishes in the discontinuity map.

In the case of the North Sea horst image, the horizontal correlation reveals a complex set of linear discontinuity events. The sedimentary layers are removed in the time slice and the faults of the original image can be related to the faults in the attribute image. However, the attribute fault indicates additional discontinuities. Are these discontinuities significant for the

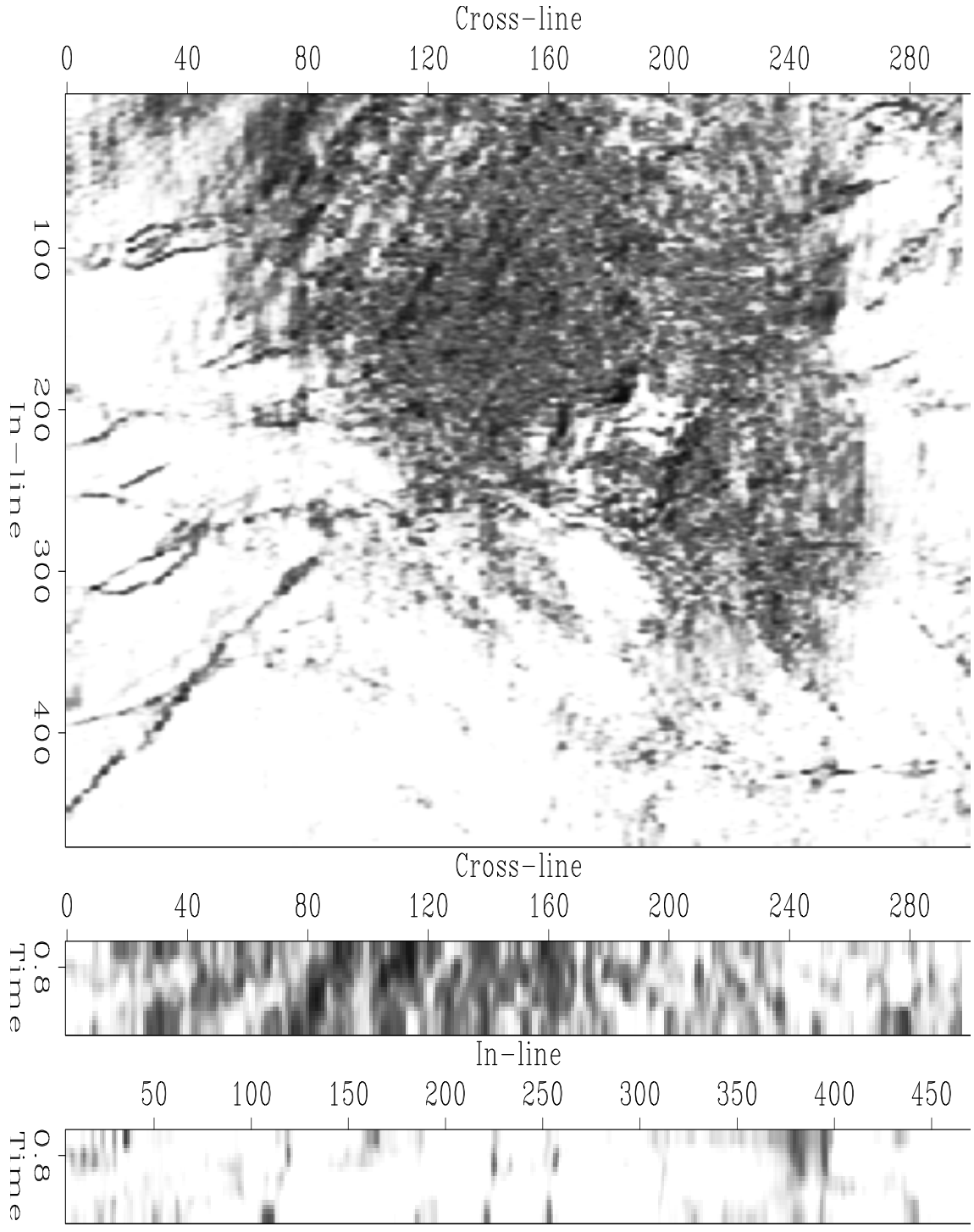

Figure 3.11: Horizontal correlator applied to salt image. The correlation coefficient of neighboring traces removes the image's sedimentary layers and delineates its faults. The simple method is remarkably successful, but does not resolve many details. paper-gulfFoltTotCorr [CR]

image's geological interpretation?

In Figure 3.12, the discontinuities contrast sharply with the surrounding sedimentary packages. However, the processes resolution is about the size of the correlation patches.

Horizontal correlation, as a semi-successful discontinuity attribute, emphasizes the importance of the within a time slice. The privileged role of the horizontal plane (and its derivatives or correlation) result from the common horizontal deposit of subsurface layers. deposited horizontally. The next section generalizes the horizontal correlation approach to images with sedimentary layers of any dip.

DISCUSSION

As expected, standard edge enhancement techniques, such as the Laplace or the gradient magnitude operator, are unable to discriminate between sedimentary layers or boundaries between layer packages, such as faults.

ACKNOWLEDGMENTS

My colleague Sergey Fomel found the Bay Area Map at a USGS site.

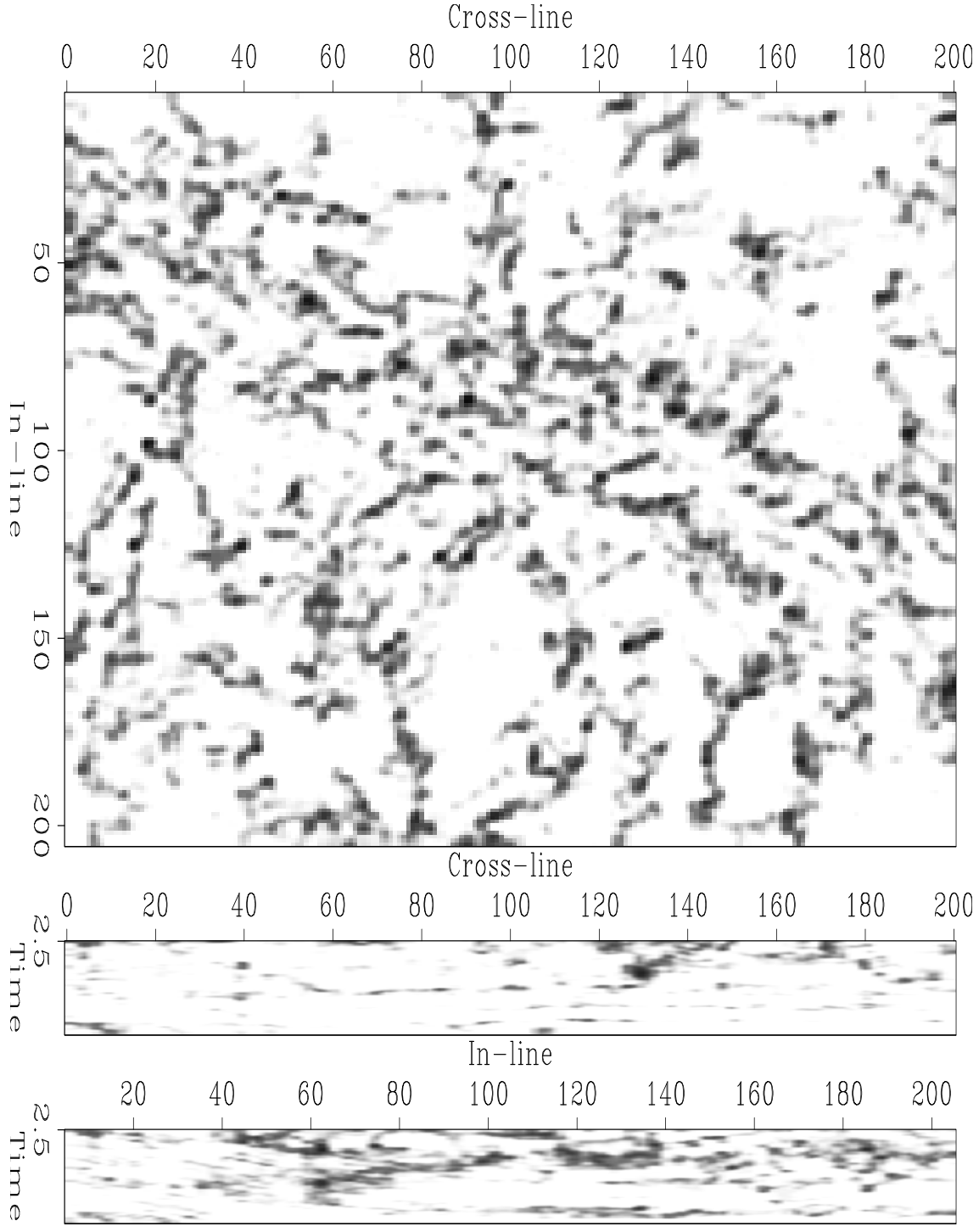

Figure 3.12: Horizontal correlator applied to horst image. In the time slice, the correlation of neighboring traces successfully removes the sedimentary layers and reveals discontinuities. However, the vertical sections show residuals of the sedimentary layers. paper-nseaFoltTotCorr [CR]

Chapter 4

Plane-wave misfit

The misfit of a local plane-wave approximation to a seismic image region – the topic of this chapter – is my most successful discontinuity attribute (Figures 4.9 and 4.10). An image location is considered free of a discontinuity if it is locally well-approximated by a plane wave. Discontinuities violate this local plane-wave assumption.

I compute the discontinuity attribute volume in a series of steps. First I partition the image volume into small sub-volumes, *patches*. Next, I estimate the best-fitting plane wave for each patch. The misfit between the plane-wave approximation and the original patch yields a local discontinuity attribute. Finally, I merge all local discontinuity patches into a single discontinuity volume comparable to the input image volume. A later chapter discusses this patching of nonstationary images in greater details.

The three-dimensional plane-wave approximation within a patch estimates the plane-wave slope by minimizing a cross-product expression of partial derivatives. The plane wave's one-dimensional waveform is approximated by stacking along the estimated plane-wave slope. Claerbout (1992a) describes an equivalent approach for dip picking in two-dimensional seismic sections. The approach can be extended to a more precise iterative estimation (Symes, 1994)

This chapter details the estimation of a best-fitting plane-wave function to a given image patch. The subsequent section discusses various measures of plane-wave fit based on either

the residual of the plane-wave dip estimation or the residual of the plane-wave waveform estimation.

PLANE-WAVE ESTIMATION

A three-dimensional plane-wave function is a projection of a one-dimensional waveform $f(s)$:

$$f(\mathbf{p}\cdot\mathbf{x}) = f(p_x x + p_y y + p_z z) \tag{4.1}$$

where $\mathbf{p}$ is normal to the constant planes. Additionally, $\mathbf{p}$ is non-zero ($||\mathbf{p}|| \neq 0$). Such a projection spreads the value $f(s)$ along the plane perpendicular to the normal direction $\mathbf{p}$. For any given point $\mathbf{x}$, the projection $s = \mathbf{p}\cdot\mathbf{x}$ yields the argument of the function f based on the geometric interpretation of the vector product.

Given an input image $g(\mathbf{x})$, plane-wave estimation finds an approximation of normal vector $\mathbf{p}$ and a waveform estimation $f(s)$. The naive formulation of the plane-wave estimation problem

$$\min_{\mathbf{p},\, f(s)} ||g(\mathbf{x}) - f(\mathbf{p}\cdot\mathbf{x})|| \text{ for all } \mathbf{x} \tag{4.2}$$

is nonlinear and highly non-convex (Symes, 1994). Alternatively, the problem can be reformulated into two well-behaved estimations. The estimation

$$\min_{\mathbf{p}} ||\mathbf{p}\times\nabla g(\mathbf{x})|| \text{ for all } \mathbf{x} \tag{4.3}$$

finds the best-fitting vector $\mathbf{p}$ that is normal to g's gradient. The subsequent estimation

$$\min_{f(s)} ||g(\mathbf{x}) - f(\mathbf{p}\cdot\mathbf{x})|| \text{ for all } \mathbf{x} \tag{4.4}$$

uses the normal $\mathbf{p}$ estimated by the first minimization to find the waveform $f(s)$.

The formulation of the cross-product minimization stems from the fact that an image volume $g(\mathbf{x})$ is a plane-wave volume with normal $\mathbf{p}$ ($\mathbf{p} \neq \mathbf{0}$), if and only if at all $\mathbf{x}$

$$\mathbf{p}\times\nabla g = \mathbf{0}. \tag{4.5}$$

If g is indeed a plane-wave function with a normal proportional to $\mathbf{p}$, then

$$g(\mathbf{x}) = f(\alpha \mathbf{p} \cdot \mathbf{x})$$

and, consequently,

$$\mathbf{p} \times \nabla g(\mathbf{x}) = \mathbf{p} \times \nabla f(\alpha\, \mathbf{p} \cdot \mathbf{x}) = \alpha f'(\alpha\, \mathbf{p} \cdot \mathbf{x}) \mathbf{p} \times \mathbf{p} = \mathbf{0}$$

Similarly, any image $g(\mathbf{x})$ that satisfies equation (4.5) can be shown to be a plane-wave volume as defined in equation (4.1).

The constraint of a nonzero normal vector $\mathbf{p} \neq 0$ can be implemented in various ways. The constraint $|\mathbf{p}| = 1$ leads to a symmetric formulation in all spatial variables x, y, and z. However, for mathematical convenience, I choose the constraint $p_z = 1$, which excludes vertical plane waves and leads to an asymmetry between the spatial variables. In practice, this particular constrain does not limit the processing of ordinary seismic images since image features are rarely vertical.

The minimization of equation (4.5) can be solved analytically. Let $\mathbf{g}_x$ be the vector of partial x-derivatives of discretized g. Each element represents the derivative at a given location $\mathbf{x}$. The vectors $\mathbf{g}_y$ and $\mathbf{g}_z$ are the corresponding partial y- and z-derivatives. Given the constraint $p_z = 1$, equation (4.5) can be expressed in matrix form as

$$\begin{pmatrix} 0 & \mathbf{g}_z \\ \mathbf{g}_z & 0 \\ \mathbf{g}_y & -\mathbf{g}_x \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} \mathbf{g}_y \\ \mathbf{g}_x \\ 0 \end{pmatrix}.$$

The minimization's corresponding normal equation is

$$\begin{pmatrix} \mathbf{g}_y^2 + \mathbf{g}_z^2 & -\mathbf{g}_x \cdot \mathbf{g}_y \\ -\mathbf{g}_x \cdot \mathbf{g}_y & \mathbf{g}_x^2 + \mathbf{g}_z^2 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} \mathbf{g}_x \cdot \mathbf{g}_z \\ \mathbf{g}_y \cdot \mathbf{g}_z \end{pmatrix}. \tag{4.6}$$

The variables

$$q_x = \frac{\mathbf{g}_x \cdot \mathbf{g}_z}{\mathbf{g}_z \cdot \mathbf{g}_z} \text{ and } q_y = \frac{\mathbf{g}_y \cdot \mathbf{g}_z}{\mathbf{g}_z \cdot \mathbf{g}_z} \tag{4.7}$$

estimate the gradient slope in the x and y direction of g. Substitution of the gradient expressions (4.7) into the normal equation (4.6) yields

$$\begin{pmatrix} q_y^2+1 & -q_x q_y \\ -q_x q_y & q_x^2+1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} q_x \\ q_y \end{pmatrix}.$$

Finally, the equation system can be solved for the x- and y-component of the plane wave's normal vector:

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \frac{1}{(q_x^2+1)(q_y^2+1)-(q_x q_y)^2} \begin{pmatrix} q_x^2+1 & q_x q_y \\ q_x q_y & q_y^2+1 \end{pmatrix} \begin{pmatrix} q_x \\ q_y \end{pmatrix}.$$

Claerbout (1992a) shows how the gradient slopes q_x and q_y can be computed from the input image using finite-difference approximations of the partial derivative expressions $\partial g/\partial x, \partial g/\partial y$, and $\partial g/\partial z$. The solution fully specifies the normal vector $\mathbf{p}$ since the vector's z component p_z is constrained to 1.

If we set the variables in the y dimension – q_y and p_y – to zero, the solution simplifies to

$$p_x = q_x = \frac{\mathbf{g}_x \cdot \mathbf{g}_z}{\mathbf{g}_z \cdot \mathbf{g}_z},$$

which is identical to Claerbout's (1992a) two-dimensional dip picking scheme. My original cross-product formulation (4.5) expresses the natural symmetry of the plane-wave detection problem that Claerbout's Lomoplan formulation lacks. However, my constraint for the nonzero normal vector ($p_z = 1$) introduces the same asymmetry later.

Stacking of the image volume $g(\mathbf{x})$ along planes orthogonal to the estimated normal vector $\mathbf{p}$ yields a first estimate of the unknown plane-wave waveform $f(s)$. Symes (1994) formulates a similar, but more sophisticated, iterative estimation scheme of the normal vector $\mathbf{p}$ and the waveform $f(s)$.

DIP MISFIT BY CROSSPRODUCT RESIDUAL

The residual of the least-squares estimation (4.5)

$$\mathbf{r_p}(\mathbf{x}) = \mathbf{p} \times \nabla g(\mathbf{x})$$

indicates how well the seismic volume is approximated by a plane wave. The computation does not explicitly involve the plane-wave waveform $f(s)$, but is the cross-product of the the local gradient ∇g and the patch's best-fitting plane-wave gradient $\mathbf{p}$. For a three-dimensional input volume, the residual of the cross-product operator expression yields a three-dimensional vector at each output location of the three-dimensional image volume.

Results

Figure 4.1 shows the cross-product output for the synthetic test case of Figure 2.4. Each component of the local residual vectors is displayed as an individual image volume. In the regions away from the fault, where the image satisfies the plane-wave assumption, the residual is zero. At the fault, the output image is non-zero and the fault is broadly outlined. Within a patch that straddles the fault, the plane-wave dip estimation encounters two contradictory plane-wave events. The least-squares solution averages the two dips so that the dip estimate does not approximate either of the two dips. Consequently, within a patch that straddles the fault, all pixels show a significant residual. The patch-wide residual reduces the resolution of the discontinuity image to the size of the patches and does not imply a meaningful interpretation of the discontinuity amplitude.

Because a three-dimensional vector field is difficult to interpret, the schemes of the two next sections summarize the information of the three volumes into a single discontinuity image.

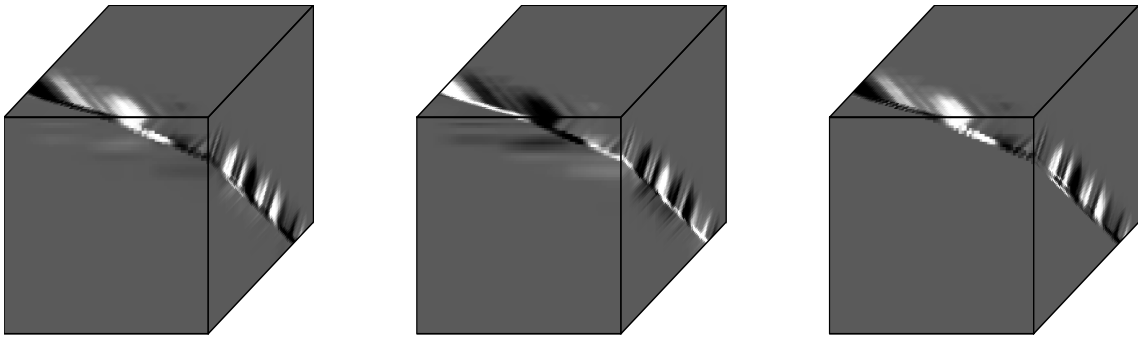

Figure 4.1: Cross-product operator applied to synthetic image. The cross-product operator applied to the synthetic fault model broadly delineates the fault in each of its three output volumes. Each output volume is a component of the cross-product vector. However, the three-dimensional output vector at each pixel of the three-dimensional image is not suited to interpretation. paper-zeroFoltDXG [ER]

NORM OF RESIDUAL

The magnitude of the three-dimensional residual vector $\mathbf{r_p}$ at each patch pixel location $\mathbf{x}$,

$$r_L(\mathbf{x}) = \sqrt{\sum_{\mathbf{p}} \mathbf{r_p}(\mathbf{x}) \cdot \mathbf{r_p}(\mathbf{x})},$$

collapses the vector field $\mathbf{r_p}$ to an image of the same size and shape as the input. The scalar r_L is zero if and only if the local $\mathbf{r_p}$ is the null vector, which in turn implies that the local gradient $\nabla g(\mathbf{x})$ is parallel to the patch's best-fitting plane-wave normal $\mathbf{p}$.

Results

The magnitude of the residuals of the synthetic image example (Figure 4.2) resembles its constituent components (Figure 4.1). Patches that contain pure plane waves are zeroed. Patches that straddle the fault and contain two plane waves of distinct dip show significant residual amplitudes. The limit of resolution of the operator is the patch and, again, the fault is broadly outlined but not pinpointed. The discontinuity amplitude does not have a simple interpretation,

except that it indicates plane-wave elements that are not well approximated by the somewhat arbitrary best-fitting local plane wave. Interestingly, the generally lower amplitudes of the horizontal plane-wave region indicate that the the best-fitting local plane waves tend to approximate the horizontal events better than the dipping ones. The side of the local cubic patch is 12 pixels long; the side of the total input cube is 80 pixels long.

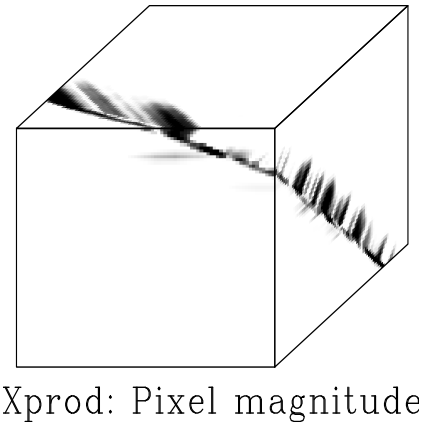

Figure 4.2: Pixel magnitude of cross-product residual of the synthetic image. The pixel magnitude of the cross-product residual of Figure 4.1 roughly delineates the fault of the input image. Again, instead of pinpointing the fault, the operator highlights any local patch that violates the plane-wave assumption. paper-zeroFoltDXGLN [ER]

Applied to the Gulf salt dome image (Figure 4.3), the magnitude of the crossproduct operator generates a noisy discontinuity map that is void of sedimentary layers but only hints at the image's faults. The attribute suppresses the sedimentary layers successfully, even the steeply dipping ones in the north-west corner. The radiating faults (e.g., the one marked Rin the original image 2.8) are depicted as linear features of high discontinuity amplitude. However, the image's general random noise obscures its faults. The internal area of the salt body does not generate high amplitudes. The pentagonoid region within the salt, the diagonally truncating fault, and the salt-sediment boundary are broadly outlined by a fuzzy zone of high discontinuity.

The interpretability of the image features suffers from its high noise-to-signal ratio that spoils its contrast and sharpness. The resolution of the radial faults is again the size of the patches. Moderate temporal smoothing might slightly improve the image without loss of resolution.

In the case of the North Sea horst image (not shown), the randomness of the time slice defeats interpretation, even though a few linear features are discernible. Disappointingly, the vertical section of the North Sea horst image depict traces of sedimentary layers. In summary,

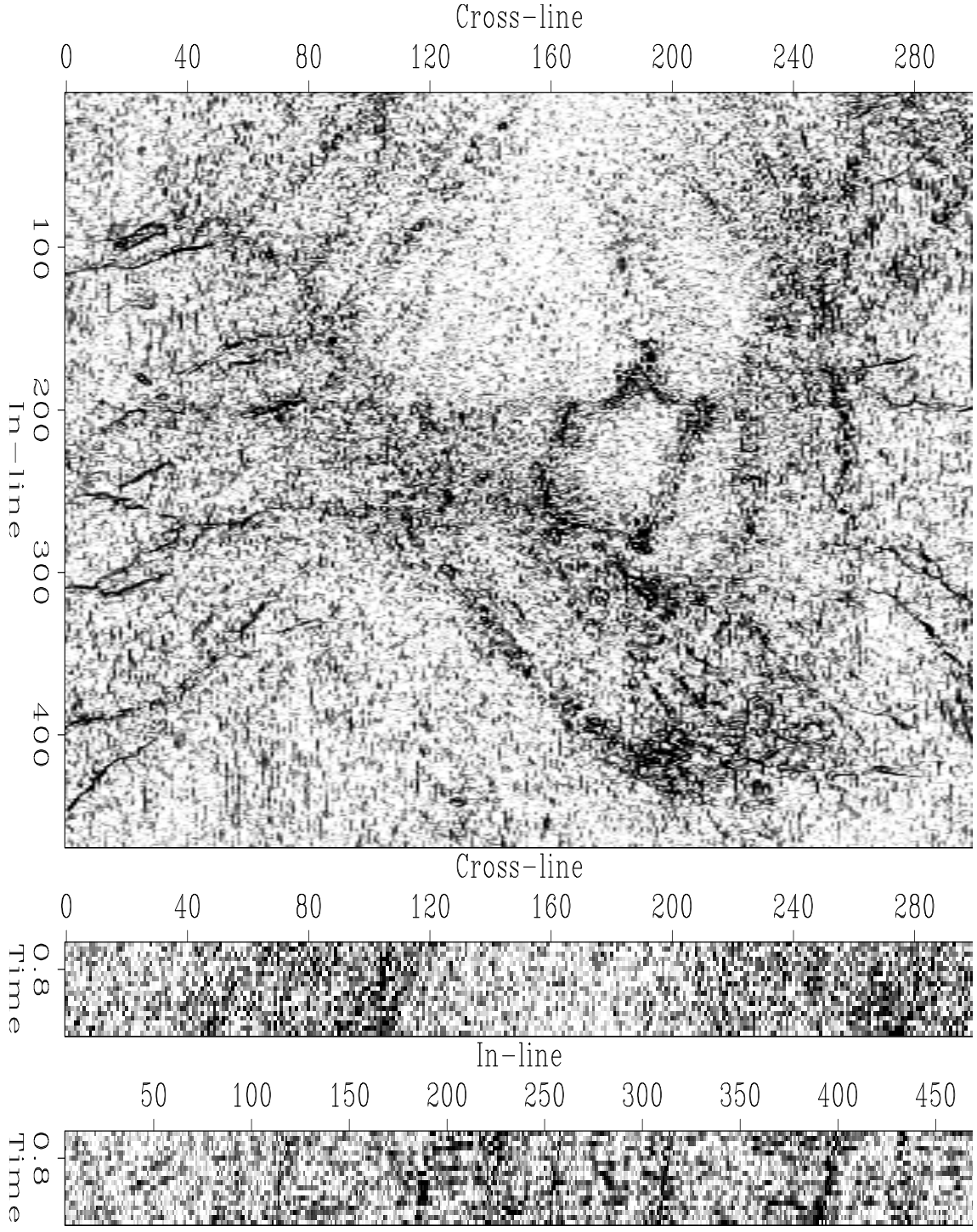

Figure 4.3: Pixel magnitude of cross-product residual of the Gulf salt image. The noisy discontinuity map successfully suppresses the original wavefield character and fuzzily indicates the image's faults. paper-gulfFoltTotDXGLN [CR]

the magnitude of the cross-product residual failed to compute an interpretable discontinuity map.

BACKPROJECTION OF RESIDUAL

The compounding of the cross-product operator (4.5) and its adjoint yields an alternative discontinuity measure, the back-projected residual:

$$r_A(\mathbf{x}) = (\mathbf{p} \times \nabla) \cdot \mathbf{r_p}(\mathbf{x}) = (\mathbf{p} \times \nabla) \cdot (\mathbf{p} \times \nabla) g(\mathbf{x})$$

The output and input image agree in size and shape. In contrast to all-positive norm expressions, the back-projected residual can be negative. Unfortunately, I do not have an insightful interpretation of the back-projected residual.

Results

The backprojected residual of the synthetic image example (Figure 4.4) resembles the individual components of the crossproduct operator. The fault is outlined but directly bordered by large residual amplitudes of the neighboring plane wave regions. Consequently, the resolution is about the size of the patch. Otherwise, the plane wave patches away from the discontinuity are zeroed.

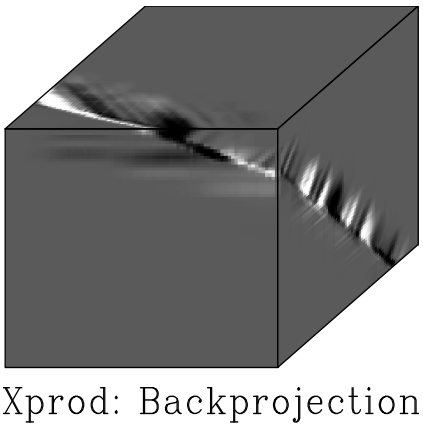

Figure 4.4: Forward and adjoint of cross-product operator applied to the synthetic image. The back-projected residual of the synthetic test image suppresses the pure plane-wave patches of the image and broadly delineates the input's fault location. paper-zeroFoltDXGAA [ER]

In the case of the Gulf salt dome image (Figure 4.5), the back-projecting crossproduct operator removes the image's sedimentary layers successfully, but the delineated faults are slightly corrupted by the image's considerable noise. Some of the noise forms short north-south streaks – best visible in the triangular area in the south-west corner – and might be due to the image's acquisition footprint. The border of the central low-amplitude region is outlined by a broad and fuzzy zone of higher average discontinuity amplitudes. The salt region shows some internal features, such as the linear north-south streaks in its center. The salt's pentagonoid region is visible but its western border is strangely distorted. The salt image's vertical north-south section shows a disappointing amount of remaining sedimentary layers. In contrast, the east-west section shows only a few weak hints of the former layering. Naturally, the near-vertical faults are more visible among the coherent layers of the north-south section than the rather incoherent east-west section.

The back-projected crossproduct operator applied to the North Sea horst image generates an extremely noisy discontinuity map. A few linear features can be seen at a grazing angle. The features include the fault marked F in the original image 2.12. The sparsity of features contrasts positively with the complexity of the same time slice when processed by competing discontinuity operators. However, the fault surfaces are not isolated in the image noise, and the vertical sections show considerable layering.

In summary, the back-projection operator highlights many interesting details that other discontinuity attributes miss. However, its overall low signal-to-noise ratio does prevent interpreters from quickly and conveniently assessing the geological information.

PLANE-WAVE MISFIT BY CORRELATION

The direct comparison between a local image patch and its best-fitting plane wave offers another alternative discontinuity attribute. How do I estimate the best-fitting plane wave $\hat{f}(\mathbf{p}\cdot\mathbf{x})$ of an image region $f(\mathbf{x})$? In the previous section, I showed how to estimate the best-fitting plane-wave normal $\mathbf{p}$ for a given image region. Given its normal estimate $\mathbf{p}$, stacking the image f along the hyper-planes orthogonal to $\mathbf{p}$ yields an estimate of the one-dimensional waveform $\hat{f}(s)$. The adjoint of the stacking procedure yields an estimate of the plane-wave

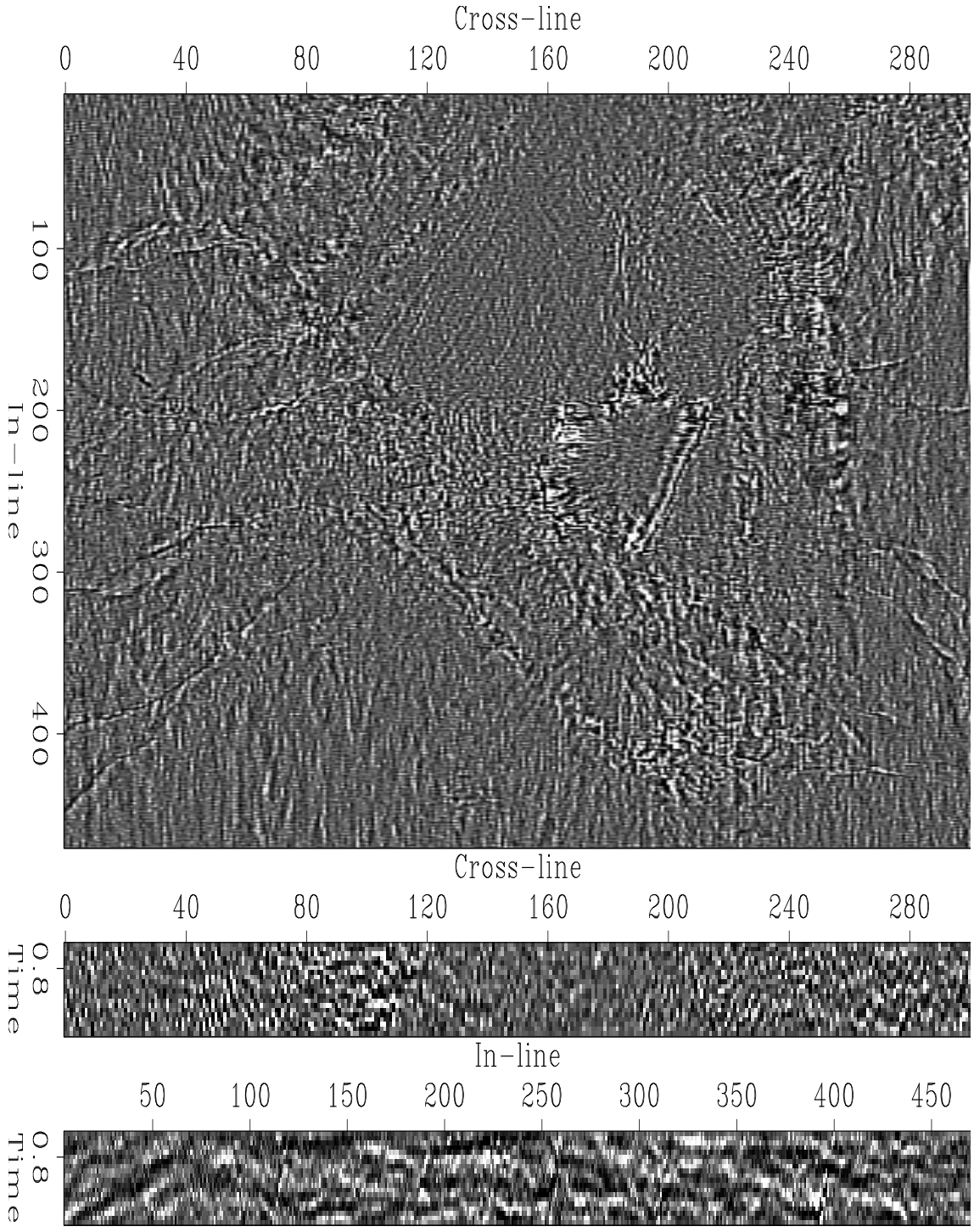

Figure 4.5: Back projection of cross-product residual of the salt image. The discontinuity map successfully suppresses the layers in the time slice and delineates the radiating faults. Overall, the image's noise makes it unattractive for interpretation. paper-gulfFoltTotDXGAA [CR]

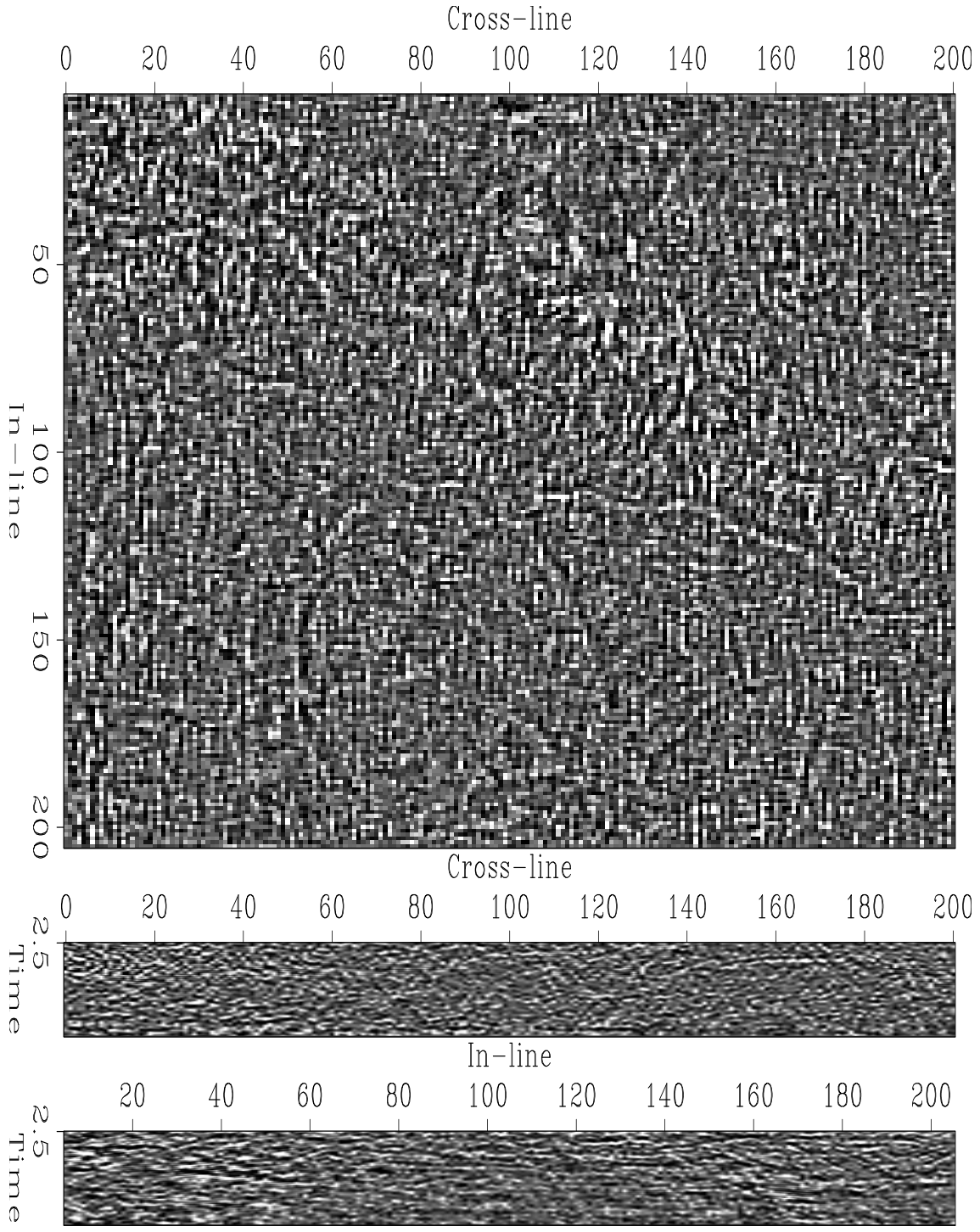

Figure 4.6: Back projection of cross-product residual of the North Sea horst image. The image is dominated by noise. At a grazing angle, the time slice depicts a few gently bending linear features (faults?). paper-nseaFoltTotDXGAA [CR]

volume $\hat{f}(\mathbf{p}\cdot\mathbf{x})$. The stacking operator and its adjoint use weights to compensate for the variable number of contributions for each waveform element.

The plane wave $\hat{f}(\mathbf{p}\cdot\mathbf{x})$ is not best-fitting the original $f(\mathbf{p}\cdot\mathbf{x})$ in a least-square sense of the expression (4.2). Instead, the normal $\mathbf{p}$ is a least-squares approximation of the true normal. The waveform is an estimate that depends on the normal $\mathbf{p}$ (rather than being estimated simultaneously). If the original image is a plane wave, my process, of course, yields the correct estimate.

The reformulation of the problem greatly simplifies the plane-wave estimation. I reformulate a highly nonlinear, non-convex cost function into a linear estimation of the plane-wave normal $\mathbf{p}$ and the linear estimation of the waveform f.

Many geophysical problems are conveniently expressed as optimization problems. Unfortunately, the computational costs of these optimization problems (often expressed by the number of iterations necessary to converge) deliver these optimal formulations often unrealistic for routine computations. I believe, a central challenge for geophysicists today is the mathematical reformulation of geophysical optimization problems into computationally manageable procedures. The crossproduct formulation of the plane-wave estimation problem is such a reformulation.

Figure 4.7 displays the composite image of the local best-fitting plane waves for the synthetic test case. I subdivide the original synthetic data of Figure 2.4 into local patches without overlap, estimate in each patch the best-fitting local plane wave, and merge the local plane waves to a composite image. The plane-wave components in the dipping and horizontal half-space are correctly estimated. Along the boundary of the two half-spaces, however, the synthetic data deviates from the single plane-wave assumption. The least-squares estimate of the dip averages the correct values of either side. The combination of stacking and its adjoint nevertheless fills each patch with a plane wave volume, which, of course, does not resemble the original patch contents of two plane waves.

To compare the best-fitting plane wave and the original local image region I correlate the patches' corresponding individual traces (the patches' individual time intervals). The discontinuity attribute divides the image into patches, estimates the best-fitting plane wave of each

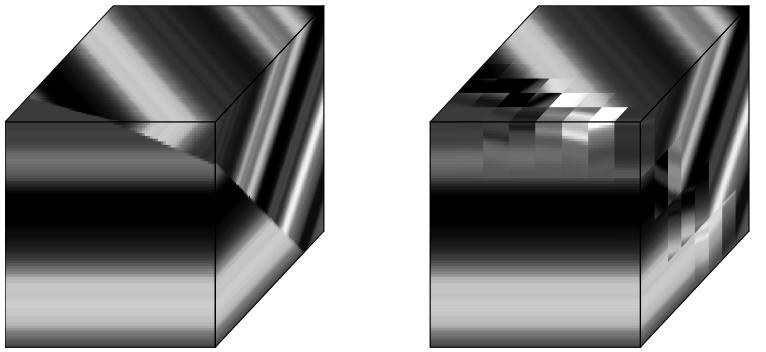

Figure 4.7: Locally best-fitting plane-wave volumes in the synthetic image 2.4. Away from the fault surface, each patch's plane wave is correctly estimated. For patches along the fault surface, the single plane-wave assumption is violated and the dip and local wavefield estimates are incorrect. paper-zeroFoltPP [ER]

patch, computes a correlation coefficient for each trace of a patch, sets all samples of a trace to the correlation coefficient, merges such patches – now indicating correlation – into a single image volume.

The correlation of two traces is

$$c(f, \hat{f}) = \frac{\sum_i f_i \hat{f}_i}{\sqrt{\sum_i f_i^2 \sum_i \hat{f}_i^2}}$$

where i loops over the samples of the one-dimensional trace f and $\hat{f}$. The coefficient is bounded by $[-1, 1]$ and yields 1 only if the two traces are identical.

In general, correlation is a robust estimate of similarity as it averages over an ensemble of samples. The discontinuity attribute could loop over all kind of subsets of an individual patch to compute correlation. The entire patch could be thought as a supertrace, which would be consistent with the plane-wave estimation of a single patch. I choose a single coefficient per trace since it averages over time samples and, thereby, looses vertical resolution, but preserves lateral resolution. Such a tradeoff seems to be well-suited for seismic data, which is usually oversampled along the time axis, but sparsely sampled in space, and which often depicts horizontal strata. Nevertheless, the overall lateral resolution of the attribute is limited by the

attributes' ability to estimate the normal and waveform at discontinuities (as described above).

As with the display of the horizontal correlation attribute, I simplify the attribute maps by linearly transforming the correlation values to non-negative numbers. The optimal correlation of 1 is displayed as 0 amplitude, the better correlated the image, the smaller the attribute value. In comparison, the correlation (3.6) in this section estimates the similarity between two one-dimensional arrays. The correlation (3.7) used in the horizontal correlation coefficient, on the other hand, estimates the similarity *among a set* of one-dimensional arrays.

Results

Overall, the correlation of the best-fitting plane wave is the my most successful discontinuity attribute. The synthetic test case shown in Figure 4.8 overall delineates the fault. At the cube's side views, the elongated events that constitute the fault indicate the attribute's loss of vertical resolution. In contrast, the fault is sharply defined in the top view. Unfortunately, the increased lateral resolution causes the fault line to be interrupted where adjacent layers lack contrast.

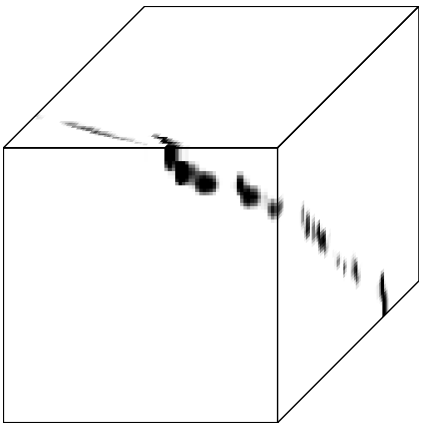

Figure 4.8: Correlation of local plane-wave estimate and original image. The fault is resolved sharply laterally, but is blurred vertically. The sharp lateral resolution leads to gaps in the fault line where the original adjacent layers lack contrast. paper-zeroFoltPPC [ER]

The first seismic test case in Figure 4.9 is a success. The wavefield character of the image due to the sedimentary layers vanished. The major faults are delineated, particular the large fault in the south-west corner. The faults are sharply resolved, and contrast well with the otherwise eventless background.

Unfortunately, the image shows a few shortcomings. In some locations, the image shows a distinct ringing that seems to be due to insufficiently suppressed, steeply dipping layers. The

salt body and its internal chaotic reflections obscure the center of the image. The pentagonoid region is not delineated, nor are the major truncating faults within the salt body. In places, the image shows some distinct north-south streaks, which I believe are due to the acquisition footprint that is also visible in the original image.

The loss of resolution and lack of faults in the vertical sections of Figure 4.9 is due to the correlation over time. The distinct border between the upper and lower data part indicates that the patch size should possibly be reduced.

Overall, the quality of the discontinuity image seems comparable to the quality of commercial discontinuity maps (Bahorich and Farmer, 1995; Luo et al., 1996). My salt dome image is a subimage of one of Bahorich's examples and the attributes can be compared directly.

The discontinuity image of Figure 4.10 suppresses the wavefield character of the original image, which was not excessive in the first place. The discontinuity map reveals a complex pattern of numerous faults and discontinuities. The discontinuities are well resolved. The discontinuity map is slightly contaminated with noise, which reduces the sharpness and contrast of the individual events and which is most-visible in the image's center. The noise might be due to events that only partially fall within the correlation's temporal averaging. The vertical section indicate that a few strong sedimentary layers at the bottom were not suppressed well.

Are the image's discontinuities truthfully delineated? The faults that I can identify in the original image all correspond to events of the discontinuity map. Some of the events – e.g., the fault marked F in the original image 2.12 – deviate from the rather continuous and smooth event that I detect in the original seismic image. Other equally strong events of the discontinuity image seem to relate to what amounts to small, hardly visible discontinuities in the original image. If my observations are correct, one may argue that the discontinuity attribute fails to distinguish between major and minor discontinuities. On the other hand, the method deserves credit for detecting such minor discontinuities.

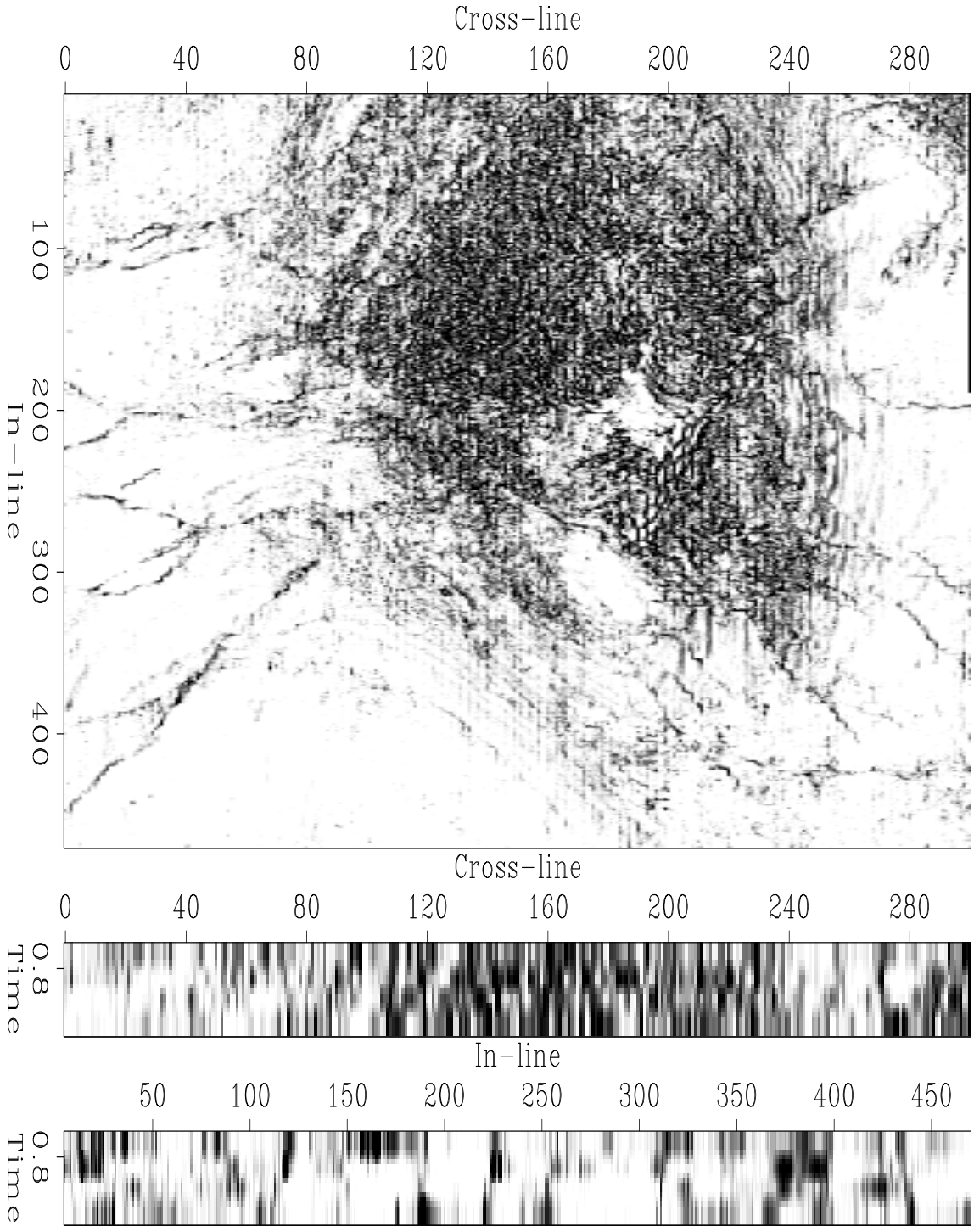

Figure 4.9: Plane-wave correlation applied to the salt image. The image's time slice successfully delineates the subsurface faults and is easily interpreted. paper-gulfFoltTotPPC [CR]

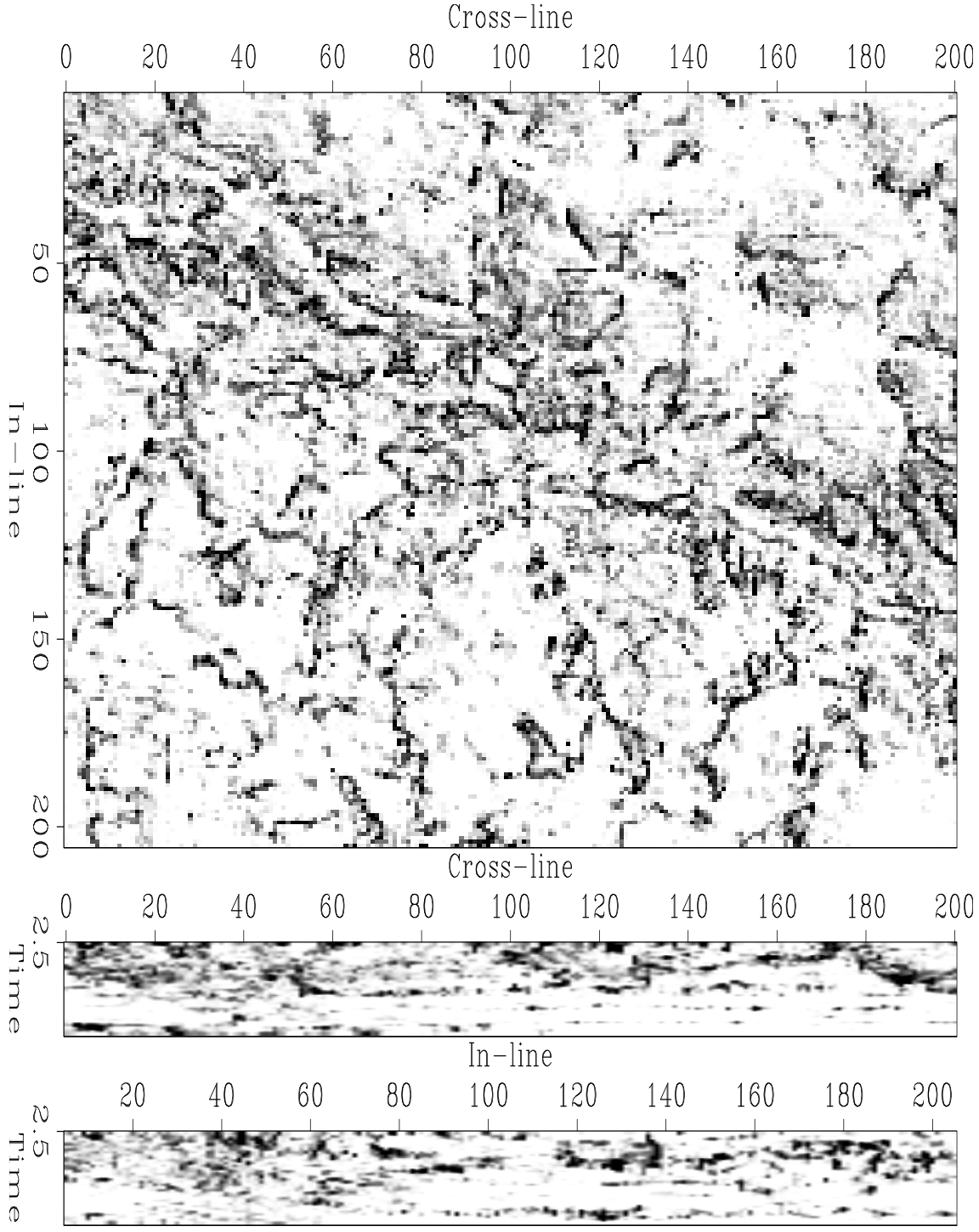

Figure 4.10: Plane-wave correlation applied to the horst image. The discontinuities of the image are delineated and the image offers a distinctively different and informative view of the subsurface. paper-nseaFoltTotPPC [CR]

DISCUSSION

The trace correlation between an image region and its best-fitting plane-wave yields an easily interpreted seismic section. The resolution of the discontinuity map is about the size of the local patch that is used to estimate the best-fitting plane wave. The quality of the discontinuity map is comparable to similar to Amoco's original coherency attribute. In comparison to the correlation, alternative misfit measures of the locally best-fitting plane wave yielded inferior discontinuity maps.

Plane-wave estimation

The plane-wave estimation by my cross-product expression is both inexpensive and elegant. I believe the formulation enables us to routinely use plane-wave estimation in a wide variety of processing applications. I am particularly interested in the fitting of the dip of seismic reflections and of well log readings. Another application is the migration of sparse seismic data using reflection dip.

ACKNOWLEDGMENTS

Jon Claerbout previously developed a two-dimensional finite-difference dip estimation scheme that turned out to be related to the three-dimensional one I discovered. Bee Bednar urged me to combine *somehow* two of Jon's operators to a three-dimensional discontinuity attribute and, thereby, prodded me into exploring the plane-wave estimation in greater detail than I first intended. Bill Symes pointed out the relationship between the cross-product operator and his Differential Semblance Optimization work.

Chapter 5

Prediction error

Prediction error is a discontinuity attribute that removes the predictable image components and reveals the unpredictable. To use prediction error as a discontinuity attribute – the original goal and starting point of my project – one has to devise a prediction-error computation that predicts and removes the plane-wave volumes of sedimentary layers but that is incapable of predicting the discontinuities.

Prediction-error filters (Claerbout, 1992a; Jain, 1989) remove the image component that is predictable as a linear combination of its neighboring values. Given a two-dimensional image $u(m,n)$ and the linear prediction estimate

$$\hat{u}(m,n) = \sum_{k \in S_x} \sum_{l \in S_x} a(k,l)\, u(m-k, n-l),$$

the prediction error is simply $\epsilon(m,n) = u(m,n) - \hat{u}(m,n)$.

The region S_x defines which neighboring values contribute to the linear prediction. Causal predictions, that involve regions of the shape shown in Figure 5.1, lead to white noise driven output images. The region is called *causal* since given a hypothetical scan from top to bottom and left to right all points of S_x lie to one-side of the predicted value $u(m,n)$.

To compute the prediction error of a given stationary image, we first find the prediction coefficients $a(k,l)$ that minimize the prediction error $\epsilon(m,n)$ for all pixels of the input image.

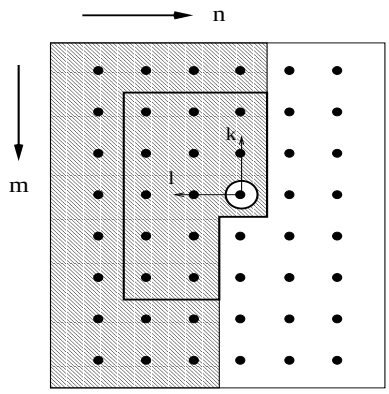

Figure 5.1: Prediction-error domain. The causal domain ensures that the output of the prediction-error filter tends to be white noise. paper-pefDomain [NR]

Once the prediction coefficients are known, the convolution

$$\epsilon(m,n) = u(m,n) - \sum_{k \in S_x} \sum_{l \in S_x} a(k,l)\, u(m-k, n-l).$$

computes the prediction error. In particular, a prediction-error filter potentially zeroes a random plane-wave, or a superposition of random plane waves, or a superposition of random constant-amplitude lines. I represent the prediction-error computation as

$$\epsilon(\mathbf{x}) = A\, g(\mathbf{x}),$$

where g is the input image, A is the prediction-error operator, and ϵ the residual prediction error.

To compute the prediction error of a nonstationary image, I, as usual, divide the image into stationary patches, compute the prediction error for each patch, and merge the patches containing the local prediction error into a single quilt. All result images of this section are smoothed along the vertical axis to suppress the prediction error's tendency to enhance high-frequency noise of the original unfiltered image.

THREE-DIMENSIONAL PREDICTION-ERROR FILTER

A three-dimensional prediction-error filter applied to a stationary image volume will tend to whiten it unconditionally. Potentially, any coherent component of a stationary image is removed. In practice, the size of the filter and the accuracy of the estimation of its coefficients limit the predictive powers of a filter and yield an image that is not entirely white. In particular, a three-dimensional prediction filter removes a plane-wave image.

Results

Figure 5.2 shows that a single three-dimensional prediction-error filter applied to the synthetic test image suppresses the plane-wave events in the patches away from the fault surface. For patches that straddle the fault, the prediction error is considerable. In these nonstationary patches, the prediction coefficients are optimized to remove plane waves of two different dip simultaneously. However, since the plane-wave events are not superimposed but adjacent, the filter is unable to predict either event perfectly. The unpredictable nature of neighboring amplitudes along the discontinuity yields particularly high amplitudes.

The horizontal events tend to be better predicted than the dipping events, probably due to round-off errors.

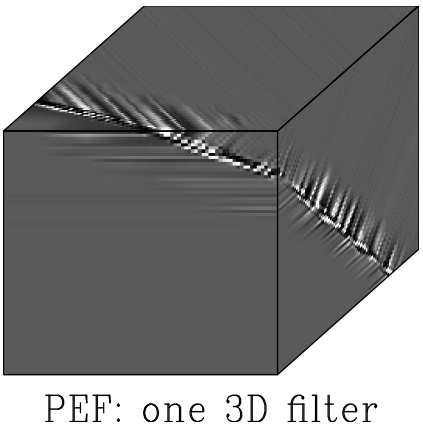

Figure 5.2: 3-D prediction-error of synthetic image. The three-dimensional prediction-error filter predicts and removes the plane-wave events in patches undisturbed by the discontinuity. The patches that straddle the discontinuity are only partially removed. paper-zeroFolt1Pef3d [ER]

Figure 5.3 shows the prediction-error attribute of the salt dome image. The image's original wavefield character is completely suppressed. The image indicates the faults, but their

exact location is obscured and interrupted by noise. Due to the noise, the image lacks sharpness and contrast. The detected faults are well resolved, but difficult to delineate. In particular, the fault marked R in the original image 2.8 is blurred at its fork. The busy region at the southern tip of the salt body hides the local faults. Within the salt, the attribute barely hints at the pentagonoid region. The salt itself is characterized by incoherent low amplitudes. The salt boundary is a zone of increased noise level. The boundary is diffuse and does not separate salt and adjacent sediments authoritatively. Overall, the attribute is too complex and noisy to benefit geological interpretation.

At a first impression, the prediction error of the North Sea horst image (Figure 5.4) appears to be white noise. Only careful inspection – especially at a grazing angle – reveals the image's complex discontinuity pattern. However, the faults are not easily delineated.

RESIDUAL OF THREE TWO-DIMENSIONAL PREDICTION-ERROR FILTERS

The three two-dimensional one-column prediction-error filters shown in Figure 5.5 predict and remove a plane-wave volume (stacked pancake model). Any other image then a plane-wave volume leads to nonzero prediction error in at least one of the three output images. In contrast to a three-dimensional vector, the three filters will not predict and remove a volume of random constant amplitude lines (spaghetti model). In that sense the three filters are a restriction of the general three-dimensional filter. Symbolically, the three fold prediction-error filtering amounts to

$$\epsilon(\mathbf{x}) = (A_x, A_y, A_z)^T g(\mathbf{x}), \tag{5.1}$$

where g is the original wavefield, A_x, A_y and A_z are the individual filter operators, and ϵ is the prediction-error vector.

The combination of the three filters generalizes the finite difference implementation of the cross-product operator of section *Plane wave misfit*. When trained at an identical plane-wave volume, the prediction-error filters and the finite difference approximations of the crossproduct operator are essentially identical. When encountering two competing plane-wave components separated by a fault, the cross-product operator estimates an average dip that leads to a large

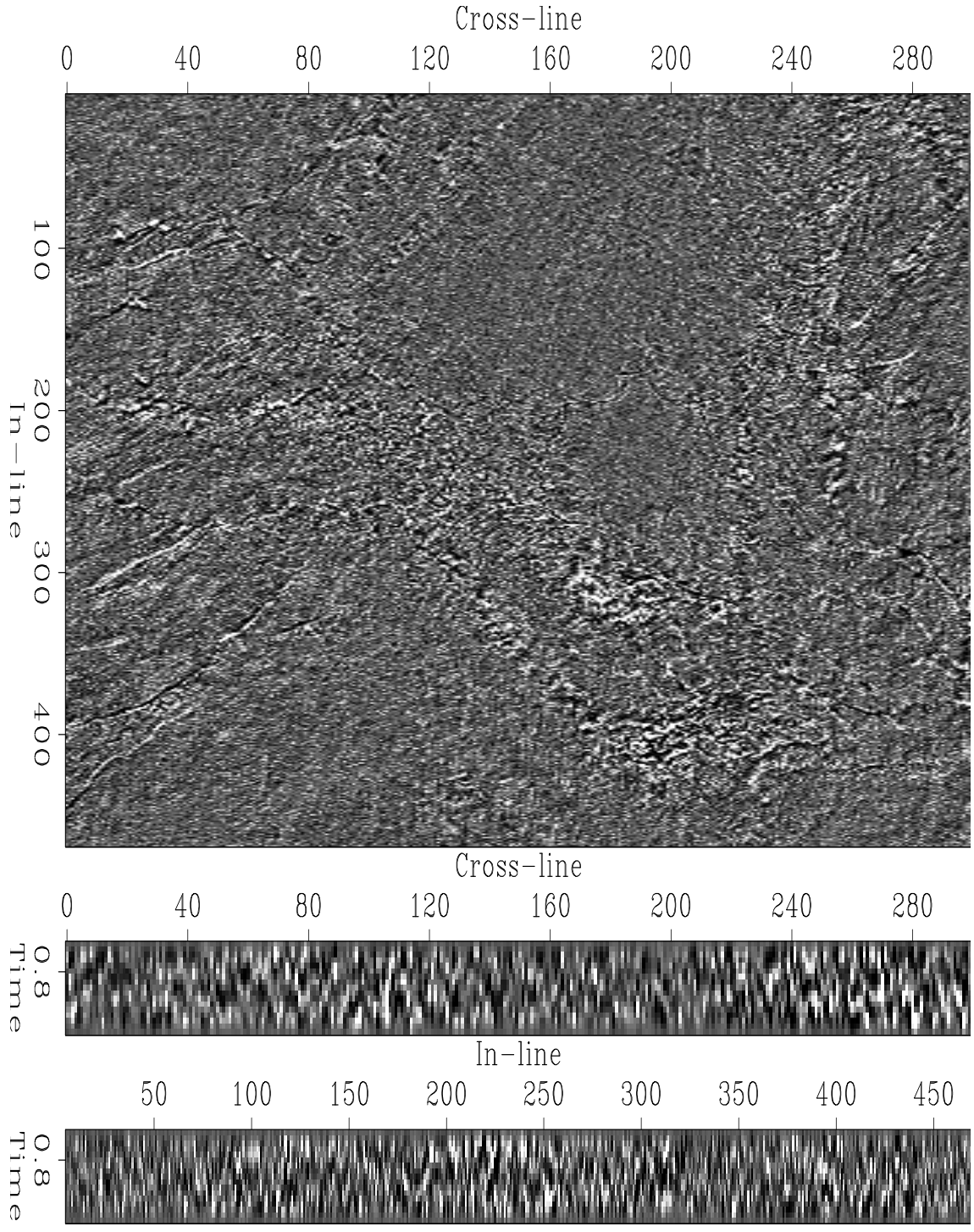

Figure 5.3: 3-D prediction-error of salt image. The attribute indicates the image's faults. In many locations, however, the attribute's noise obscures the fault features. paper-gulfFoltTot1Pef3d [CR]

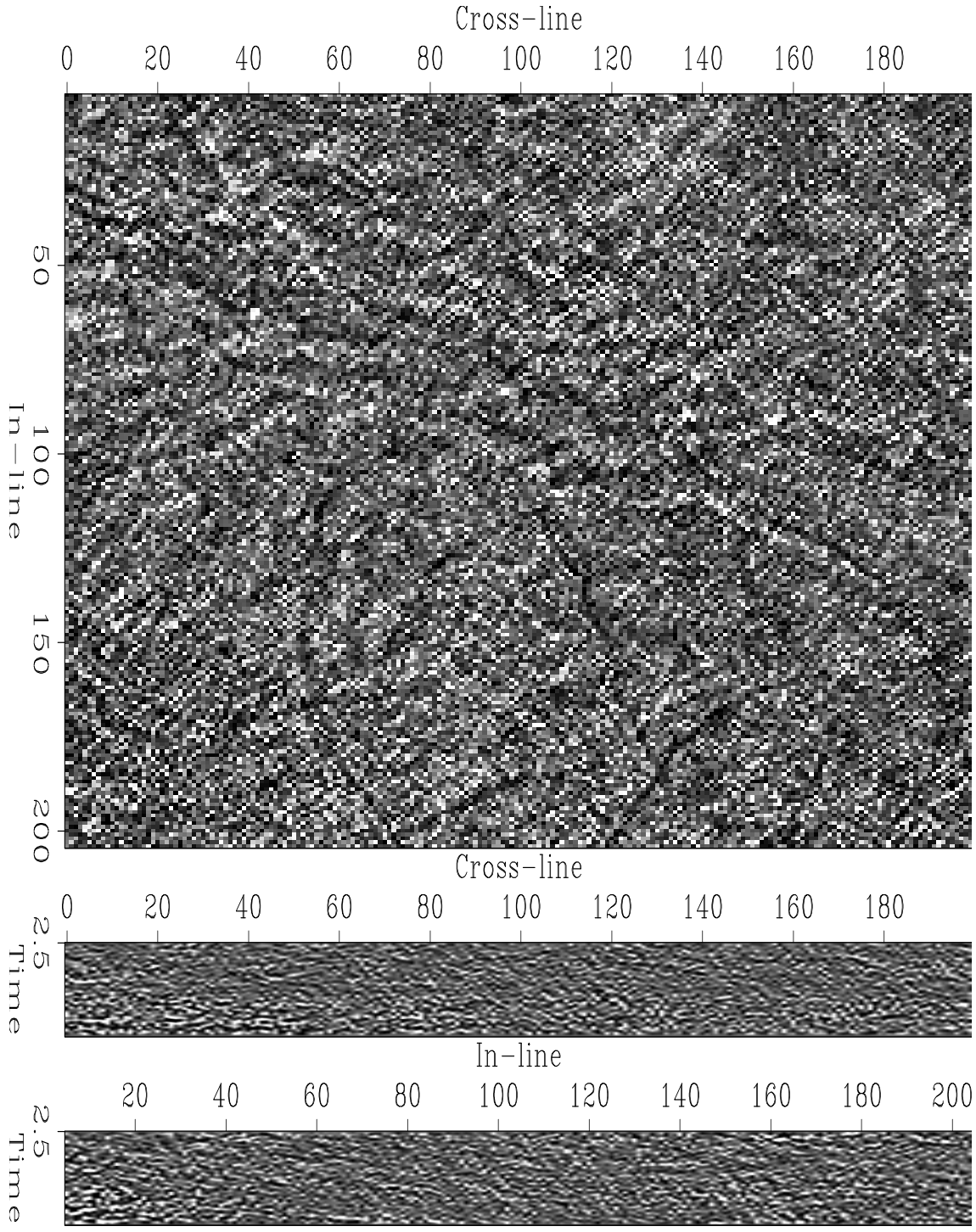

Figure 5.4: 3-D prediction-error of horst image. Among the noise, a pattern of linear events indicate the location of discontinuities to a careful observer. Unfortunately, the attribute image lacks clarity. paper-nseaFoltTot1Pef3d [CR]

residual of the crossproduct everywhere. Similarly, the three prediction-error filters adjust their coefficients to reject both plane waves. Since the plane waves are not superposed, however, the prediction error is significant everywhere.

Results

Figure 5.6 shows the prediction error of the three two-dimensional filters applied to three copies of the synthetic test case. In all three images, the fault is delineated, but blurred. In patches away from the fault, the filters encounter and remove single plane wave events. In the patches that straddle the discontinuity, the filters encounter two plane wave volumes of distinct dip. Since the filters are limited to the removal of a single plane wave, each filter generates a large prediction error in those patches. The three-dimensional vector-field format is, of course, unsuitable for human interpretation. The next sections suggest various methods that unify the vector field into a single scalar subsurface map.

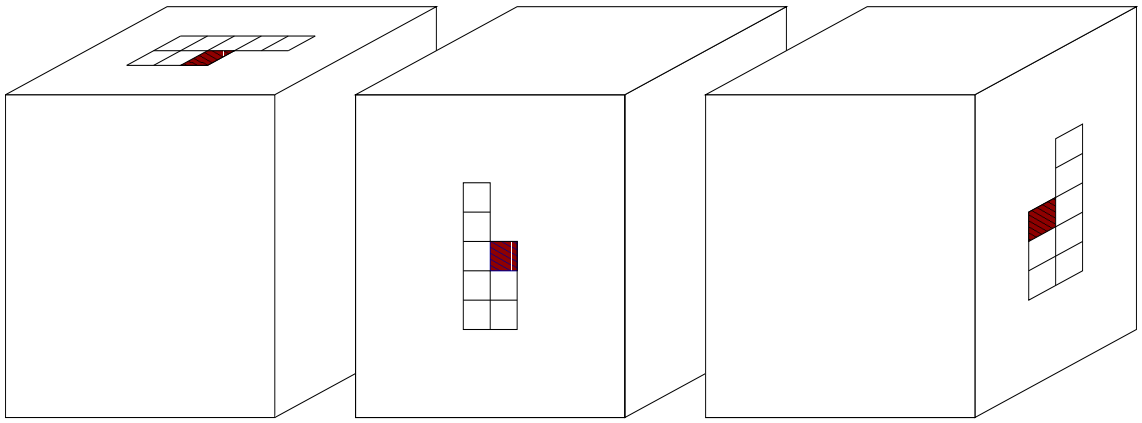

Figure 5.5: Three two-dimensional prediction-error filters. Each of three copies of the synthetic test case is filtered by one of the two-dimensional prediction-error filters of Figure 5.5. All three prediction-error volumes are zero, if and only if the input image is a single plane-wave. paper-rayab3Doper [NR]

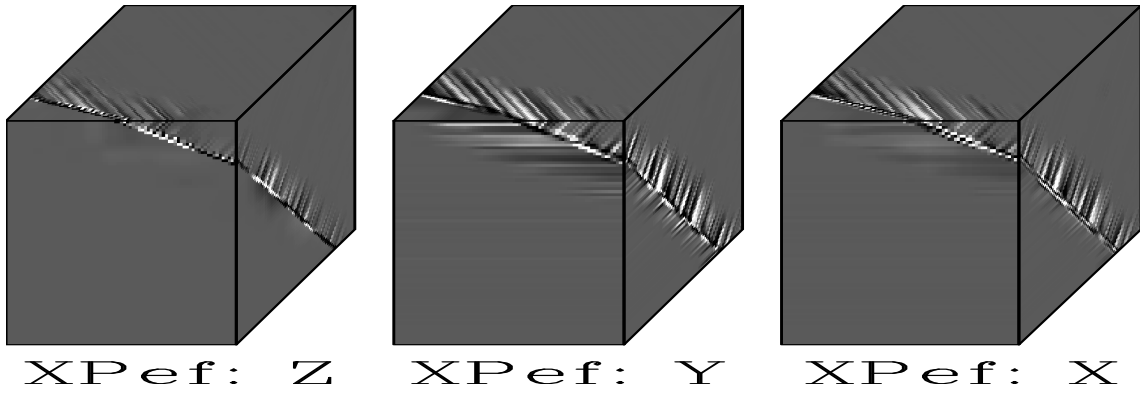

Figure 5.6: Three 2-D prediction-error of synthetic image. In all three volumes, the prediction error delineates the fault. The fault image is blurred, since the local filter is limited to removing only a single plane wave perfectly. The three-dimensional vector field is, of course, unsuitable for human interpretation and, therefore, requires further processing. paper-zeroFoltXPef [ER]

NORM OF RESIDUAL

The norm of the prediction-error vector $\epsilon(\mathbf{x})$ of expression (5.1) collapses the three image to one:

$$|\epsilon(\mathbf{x})| = |(A_x, A_y, A_z)^T g(\mathbf{x})|.$$

Results

Figure 5.7 shows that the norm of the three prediction errors broadly delineates the fault in the synthetic case. Again the patches that contain the discontinuity show large prediction errors.

The seismic test image of the Gulf salt dome indicates major fault lines as fuzzy zones of heightened prediction error (Figure 5.8). However, the overall image is noisy and the discontinuities are not pinpointed nor well contrasted from their surroundings. The sedimentary layering is completely suppressed. The salt gives rise to a region of low but noisy prediction error. Half of the pentagonoid region within the salt is clearly delineated; the other half is not. Additionally, the prediction-error attribute reveals a few features of the salt boundary that were obscured in other attribute images. For example, the long north-south fault at the eastern edge

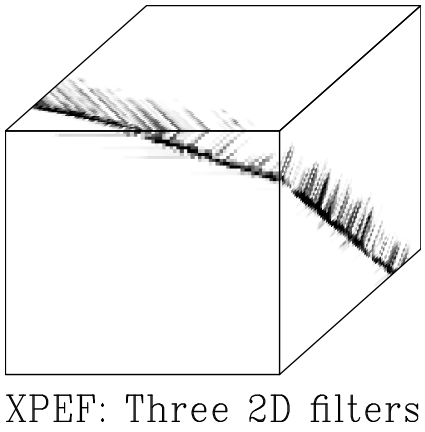

Figure 5.7: Magnitude of three 2-D prediction error of synthetic image. The fault is broadly outlined by patches of considerable prediction error. paper-zeroFoltXPefLN [ER]

of the image. The North Sea horst image (not shown) similarly hints at its faults without differentiating and resolving them sufficiently. The three two-dimensional prediction error does not yield a convenient discontinuity image.

BACKPROJECTION OF RESIDUAL

An alternative method to combine the vector field $\epsilon(\mathbf{x})$ of expression (5.1) to a scalar field is the combination of the forward and adjoint of the prediction-error operator:

$$\epsilon(\mathbf{x}) = (A_x^T, A_y^T, A_z^T) \cdot (A_x, A_y, A_z)^T g(\mathbf{x}).$$

The space of the output ϵ is equal the space of the input g. Unfortunately, I do not know a insightful interpretation of the procedure. If prediction-error filter is similar to a derivative operator, the combination of forward and adjoint corresponds to a sequence of two derivative operations.

Results

Figure 5.9 shows the discontinuity map of the synthetic image computed by the combination of the forward and adjoint of the prediction-error operator. The plane-wave patches are zeroed. The fault is delineated. The residual due to the nonstationarity of the discontinuous patches

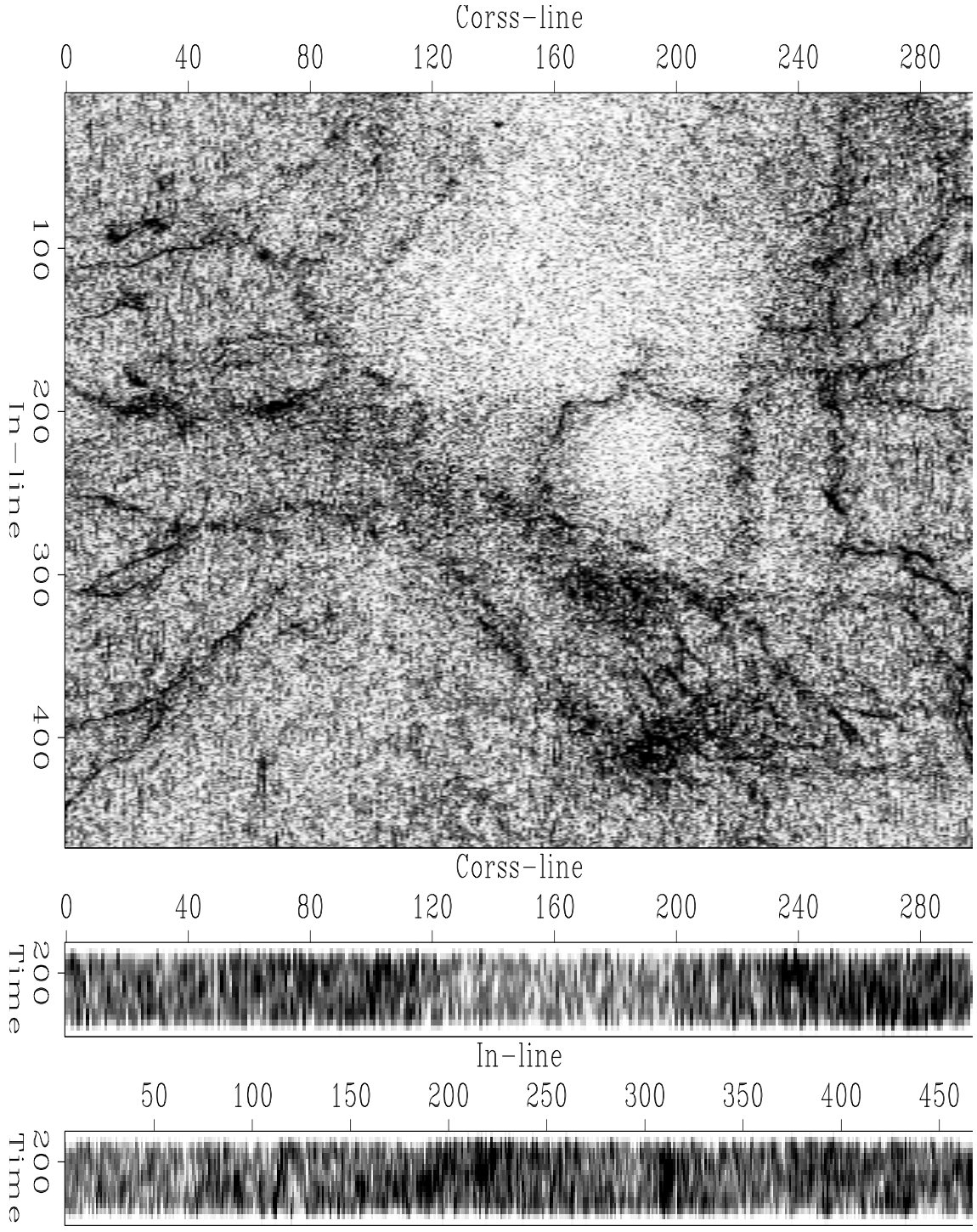

Figure 5.8: Magnitude of three 2-D prediction error of salt image. Broad, fuzzy zones indicate the image's major faults among a noisy background. paper-gulfFoltTotXPefLN [CR]

blurs the discontinuity. However, the operator seems to pinpoint the fault better than the other operators.

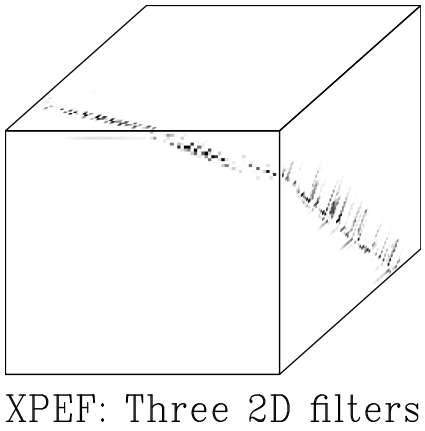

Figure 5.9: Forward and adjoint of three 2-D prediction-error filters applied to synthetic test case. The output delineates the fault, but surrounds it with some blur. paper-zeroFoltXPefAA [ER]

The back-projected prediction error suppresses most of the wavefield character of the Gulf salt dome image (Figure 5.10. Some ringing within the salt body are due to the salt's weak original internal reflections. The salt's radial faults are mostly delineated. Especially, the fault marked R in the original image 2.8 appears in great details. Unfortunately, the image is noisy and other faults are slightly obscured. The salt body itself clearly delineates the pentagonoid region. The salt boundary is a broad zone of increased prediction error. The image clearly suffers from the visible presence of the north-south streaks that might be acquisition footprint. In summary, the discontinuity attribute shows many subtle details, however, it is too noisy to be quickly and reliably interpreted.

Figure 5.11 fails to indicate the location of the major faults, which exist in the original image. The overall wavefield character of the image is replaced by a white noise character. Despite temporal smoothing, the discontinuity image only hints at some of its major faults.

DISCUSSION

Contrary to my initial theoretical expectation, prediction error is not a reliable discontinuity attribute. In particular, I expected the combination of the three two-dimensional filters to remove any plane wave and to yield a sharp prediction error where they encountered anything but a plane wave. I was naively wrong on two accounts.

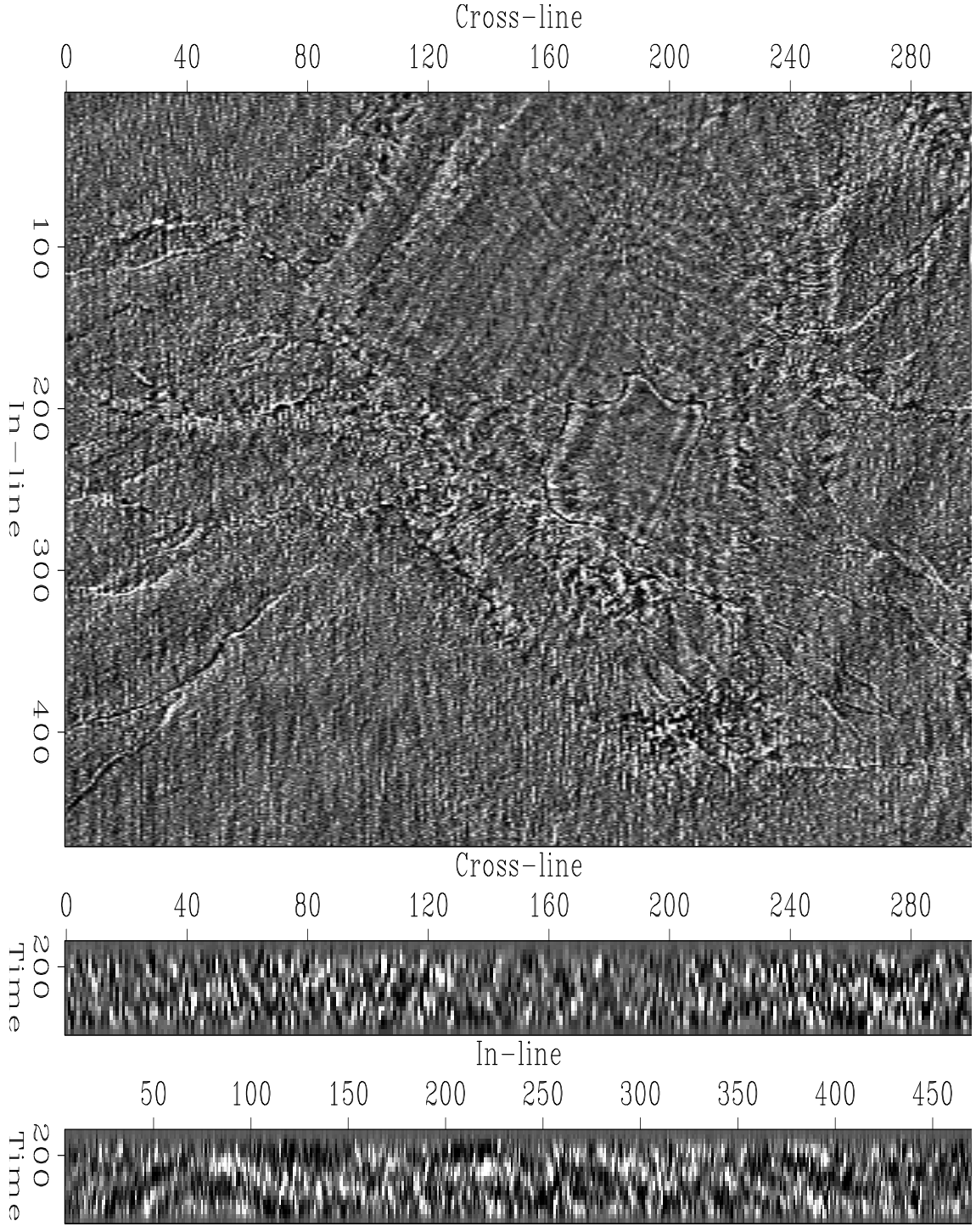

Figure 5.10: Forward and adjoint of three 2-D prediction error of salt image. The image delineates the radiating faults, however, the attributes noise level prevents the quick and reliable interpretation of the image. paper-gulfFoltTotXPefAA [CR]

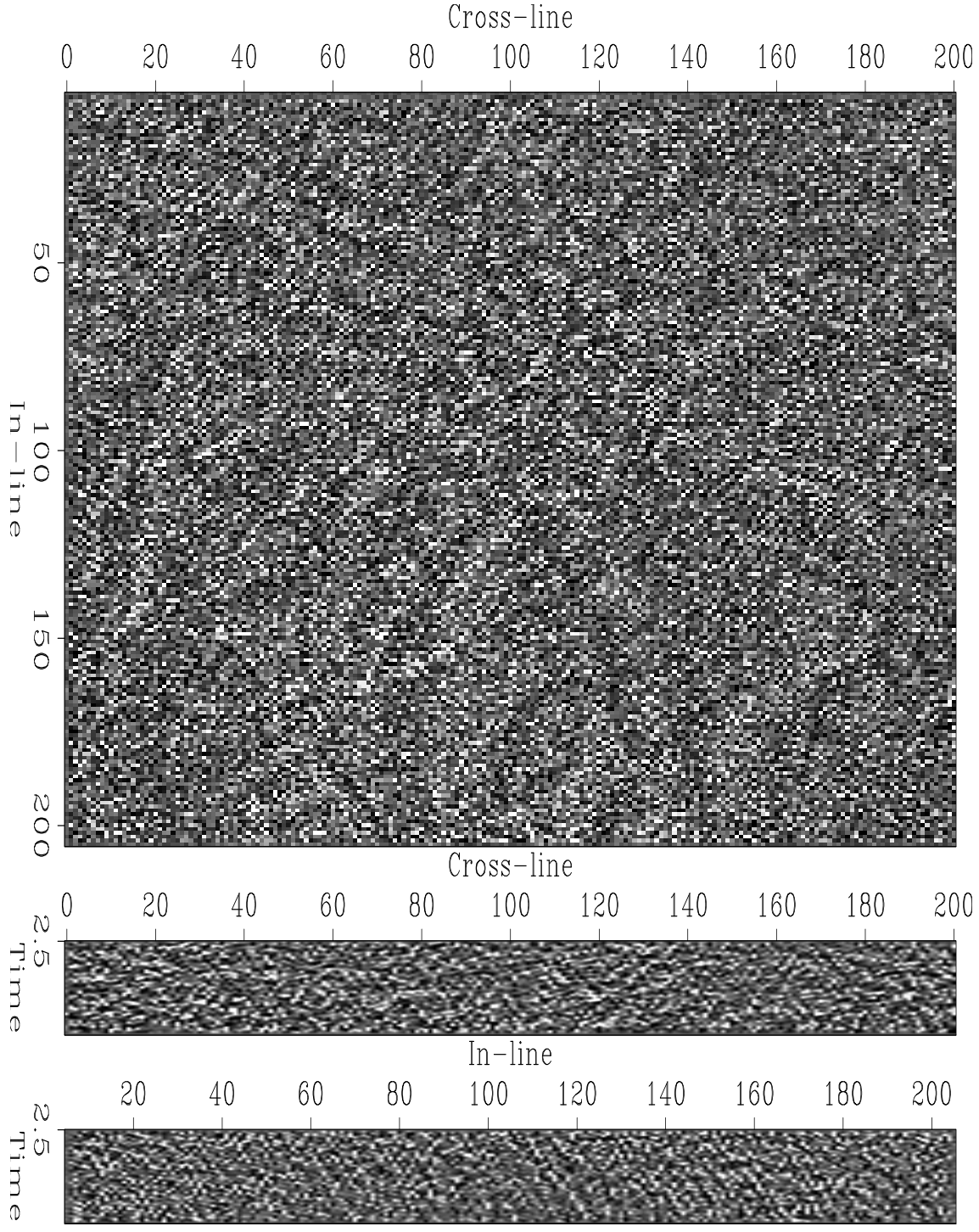

Figure 5.11: Forward and adjoint of three 2-D prediction-error filters. The discontinuity image appears noisy and fails to delineate any image features clearly. paper-nseaFoltTotXPefAA [CR]

First, I failed to realize that prediction error indicates patches that contain a discontinuity but does not pinpoint the discontinuity within the patch. I disregarded the fundamental difference between the prediction of two superposed plane waves and two adjacent plane waves. Patching breaks the initial nonstationary image into small patches. Patches that contain discontinuities are, however, still nonstationary since they contain two adjacent plane-wave volumes. In general, a single prediction-error filter cannot perfectly predict and remove the statistically distinct image regions separated by the discontinuity. Instead, the filter generates considerable error amplitudes anywhere in the patch. The resolution of a prediction-error map is consequently its patch size. For example, the blurred fault of the synthetic image cases of Figures 4.4 and 5.7 illustrate the resolution loss. In this figure the patch size is about an eighth of the image size. Similarly, the plane-wave misfit of the previous chapter detects nonstationary patches that are not approximated by a single plane-wave.

Second, I incorrectly assumed that faults were sharp pixel-to-pixel boundaries, as I stylized them in the synthetic example. Instead, image processing is often unable to deliver sharp images of discontinuities. Such smooth transition zones do not cause large prediction errors. In particular, if the size of a fault zone is similar to the size of patch, the filter will be able to predict and remove components of the fault.

The patch size has to be large enough for the reliable estimation of of the prediction-error filter coefficients. On the other hand, the patch has to be small enough to contain stationary data in most cases and to yield a high-resolution output image. For the various examples, a typical patch size is $(nx, ny, nz) = (3, 3, 10)$.

In the various test cases, the norm of the three 2-D prediction-error filters broadly delineates seismic discontinuities as fuzzy lines (Figure 5.8). In contrast, the combination of the forward and adjoint of the prediction-error filter pinpoints the discontinuities better. It even successfully highlights features within the central salt region (Figure 5.10).

In general, I found prediction error an unreliable and noisy discontinuity attribute. Prediction-error fails to detect smooth fault zones. However, prediction error delineates sharp faults. The discontinuity maps are intricate and reveal details in regions with little amplitude contrast. However, prediction-error maps are obscured by noise and are neither quickly and nor easily

interpreted.

ACKNOWLEDGMENTS

Jon Claerbout previously developed a two-dimensional and three-dimensional prediction-error scheme for the annihilation of plane waves. I used many of his excellent Fortran programs (or modified versions of them), until I changed to the Java programming language.

Chapter 6

Processing local-stationary data

A process or image is called stationary, if the statistics – mean, variance, spectrum – of any subset accurately describe the statistics of the entire data. Stationarity is so helpful that scientists assume it, even when a given data is only approximately stationary. Unfortunately, many interesting things in life are nonstationary and predictions and prejudices based on small subsets are notoriously unreliable.

Stationarity and nonstationarity are well-known properties in statistics and image processing (Leon-Garcia, 1994). Gabor's windowed Fourier transform (Gabor, 1946) and the later short-time Fourier transform (Oppenheim and Schafer, 1989) are early techniques that separate nonstationary images into stationary components. The components compactly localize events simultaneously in the time and frequency domain. In recent years, the time-frequency analysis and its applications of image compression, edge and feature detection, and texture analysis was considerably reformulated by multiresolution analysis, pyramid algorithms, and particularly wavelet transforms (Castleman, 1996). Local stationarity is a special case of nonstationarity: A local-stationary data set can be split into smaller stationary subsets (Castleman, 1996).

In this article, I first briefly define and illustrate stationarity and local-stationarity. Then I describe an algorithm that processes local-stationary data. by patching. The approach generalizes Claerbout's (1992a) patching method and encapsulate it into a set of easy-to-use,

object-oriented classes.

Definition of stationarity

A random process $y(\mathbf{x})$ is strict-sense stationary if the joint distribution of any set of samples does not depend on the sample's placement. Consequently, first order cumulative distribution functions, e.g., mean and variance, of $y(\mathbf{x})$ are constant. Furthermore, second order cumulative distribution functions (such as autocorrelation and autocovariance) depend only on the distance in placement, $\mathbf{x_2} - \mathbf{x_1}$. For example, a Gaussian process is strict-sense stationary since it is completely specified by its mean and covariance function.

If the mean is constant and the autocovariance is a function that depends only on the distance in placement, then we call the data wide-sense stationary or simply stationary. Strict-sense stationary implies wide-sense stationary.

Examples of nonstationarity

A typical seismic trace, as shown in Figure 6.1, is nonstationary since it includes parts of low and high variance, albeit its mean is approximately zero. A shot gather, as shown in Figure 6.2, is nonstationary since each individual trace is usually nonstationary. Furthermore, the hyperbolic shape of the reflection events causes spatial spectral variations.

Many synthetic data sets are stationary since they are generated by simple statistical models. In practice, data sets are often nonstationary. However, the decision if a data set is assumed stationary or nonstationary depends on an application's tolerance for deviations from strict stationarity.

MATHEMATICS

A local-stationary data set is a nonstationary data set that consists of small stationary regions. For example the shot gather of Figure 6.2 shows approximately constant dips in small local

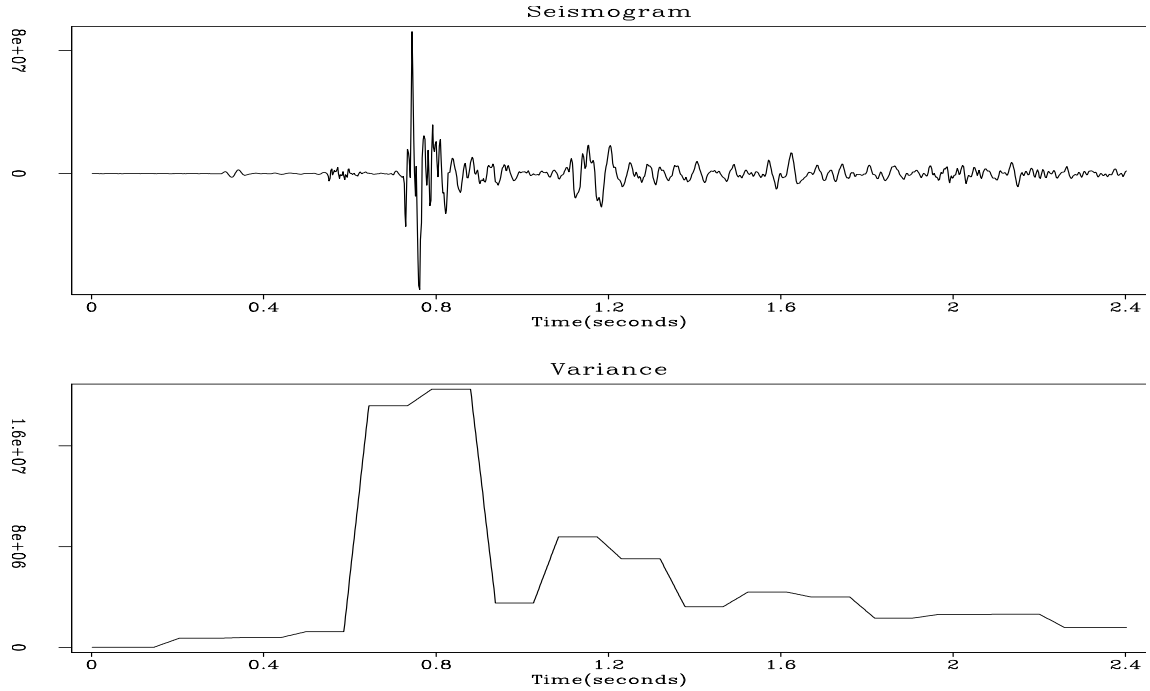

Figure 6.1: Nonstationary seismogram. The seismogram in the top panel has three distinct regions: The relatively quiet period before the event, the event, and the ringing of the coda. The bottom panel displays an estimate of the signal's local variance. paper-nStatVar [ER]

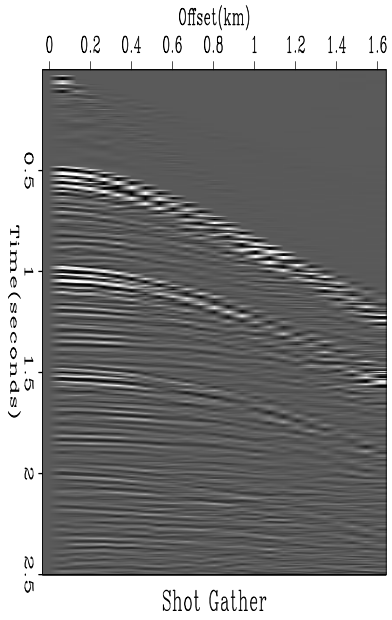

Figure 6.2: Shot gather. Shot gathers are nonstationary, because they show regions of distinct variance: The early quite zone, the hyperbolic reflections and the noisy coda at the bottom. paper-gather [ER]

data patches, albeit the dip varies considerably over the entire image. Local-stationarity suggests to split an image into individual stationary subsets. Each subset can then be treated as a stationary data with its individual statistical characterization. Finally, the individually treated subsets are merged into a single result image (Castleman, 1996; Claerbout, 1997). My object-oriented implementation of the patching approach hides all patching details from the programmer. Additionally, my implementation offers the programmer a choice of specifying the patch size and overlap of either the input or output patches.

Splitting and merging

As illustrated by Figure 6.3, the local-stationary algorithm splits the given nonstationary data set $\mathbf{d}$ into predefined stationary patches $\mathbf{d}_i$:

$$\begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{d}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} \mathbf{d}. \tag{6.1}$$

Each patch is then processed individually by a local stationary operator $\mathbf{F}_i : \Omega_{d_i} \rightarrow \Omega_{m_i}$:

$$\begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{F}_1 & & \\ & \mathbf{F}_2 & \\ & & \mathbf{F}_3 \end{bmatrix} \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{d}_3 \end{bmatrix}.$$

Finally, an interpolation operator $\mathbf{Q} = [\mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3]$ combines the processed patches $\mathbf{m}_i$ to a single output quilt $\mathbf{m}$:

$$\mathbf{m} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 & \mathbf{Q}_3 \end{bmatrix} \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix}. \tag{6.2}$$

In summary, the patching algorithm is a compounding of windowing $\mathbf{P}^T = [\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3]$, local imaging $\mathbf{F} = Diag(\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3)$, and merging $\mathbf{Q}^T = [\mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3]$:

$$\mathbf{m} = \mathbf{Q}^T \mathbf{F} \mathbf{P} \mathbf{d}.$$

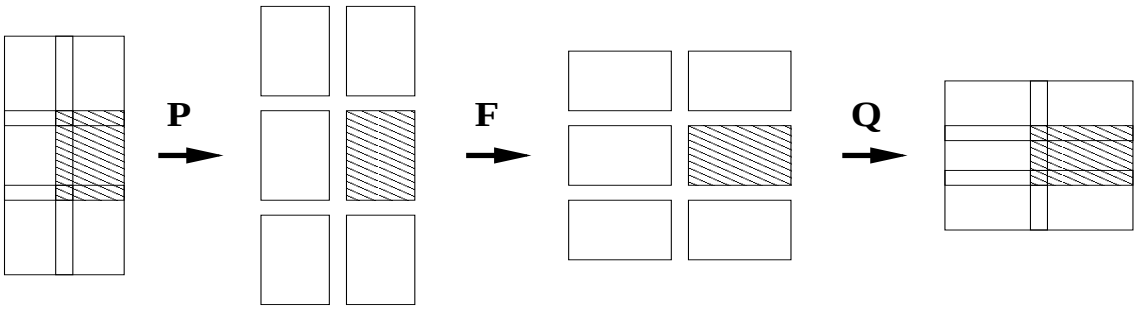

Figure 6.3: Patching scheme of local-stationary data. The algorithm extracts data patches from the data quilt. It images each patch by the corresponding stationary operator. Finally, it merges the individual patches to a single output quilt. The isolation of the individual patches makes this algorithm suitable for nonlinear, data-dependent operators. paper-quiltPatch [NR]

This algorithm applies each stationary operator exclusively to its appropriate patch. The individual patches can conveniently be used to estimate the desired operator, should it be dependent on its stationary input data. If, for example, the operators $\mathbf{F}_i$ are to deconvolve an unknown local blur, the isolated stationary patches conveniently permit the estimation of the optimal coefficients for the local deconvolution filter. However, this algorithm has to manipulate the space information of its various patches and quilts carefully: a task, the vector spaces of my Java software library *Jest* excel at.

If the operator $\mathbf{F}$ has identical domain and range, the initial windowing step (6.1) and the final merging step (6.2) could be implemented by adjoint operators: a column of truncation operators $\mathbf{P}_i$ and a row of zero padding operators $\mathbf{Q}_i = \mathbf{P}_i^T$. In general, the internal stationary operators $\mathbf{F}_i$, however, causes the range and domain of the splitting and interpolation operation to differ. Furthermore, a row of zero-padding operators amounts to stacking of the individual patches into the output quilt. To avoid visible boundaries at the edges of the patches, we usually prefer more sophisticated interpolation schemes.

Interpolation

If done carelessly, the merging of individually processed patches in equation (6.2) can cause visible patch boundaries in the final image. I implemented Claerbout's(1994) interpolation scheme, which linearly interpolates overlapping patches.

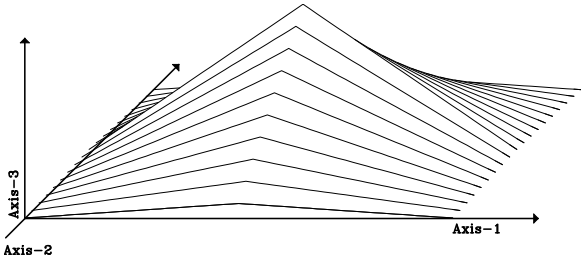

Figure 6.4: Patch weight. The weight function is applied to individual patches to prepare the patch for the linear interpolation of neighboring, overlapping patches. paper-patchwt [ER]

Interpolation requires the application of weights to each data element. The weighting is easily implemented as the element-wise scaling of two vectors: one contains the data that is to be interpolated, the other one each element's corresponding weight. Algebraically, weighting of a vector amounts to a matrix multiplication by a diagonal matrix . The diagonal elements are the elements of the weight vector.

To interpolate the area of overlap linearly, each patch $\mathbf{m}_i$ is weighted by a patch $\mathbf{w}_i$ of identical size but tent-like amplitudes (Figure 6.4). Equation (6.3) expresses the weighting as the multiplications of the diagonal matrices $\mathbf{W}_i$. After zero padding and stacking the weighted patches by the row of $\mathbf{Q}_i$ operators, the operator $\mathbf{W}_m$ is another weighting operator that compensates for the variable amount of contributions every output element received:

$$\mathbf{m} = \mathbf{W}_m \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 & \mathbf{Q}_3 \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 & & \\ & \mathbf{W}_2 & \\ & & \mathbf{W}_3 \end{bmatrix} \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{bmatrix} \tag{6.3}$$

How do we compute the operator $\mathbf{W}_m$? The same row of zero-padding operators $\mathbf{Q}_i$ that add the weighted patches $\mathbf{m}_i$ in equation (6.3) adds the patch weights into a data quilt $\hat{\mathbf{w}}_m$.

$$\hat{\mathbf{w}}_m = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 & \mathbf{Q}_3 \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 & & \\ & \mathbf{W}_2 & \\ & & \mathbf{W}_3 \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \end{bmatrix} \tag{6.4}$$

The elements of $\hat{\mathbf{w}}_m$ measure the weighted contributions each quilt element received. To balance the variable number of contributions, each element of the output quilt is scaled by the inverse of the corresponding element of the weight quilt $\hat{\mathbf{w}}_m$. This final weighting ensures

that all weights applied to a single output sample sum up to one. Algebraically, the weighting is again expressed as a matrix multiplication by a diagonal matrix $\mathbf{W}_m$ in equation (6.3). The weights $\mathbf{W}_i$ and $\mathbf{W}_m$ can be interpreted as a heuristic preconditioning by row and column weighting (Golub and Van Loan, 1989).

A user can modify the interpolation by supplying any alternative weight function $\mathbf{w}_i$ for the individual patches. The algorithm will apply the correct corresponding weight operator $\mathbf{W}_m$. Figure 6.5 shows the weight vector $\hat{\mathbf{w}}_m$ that corresponds to four slightly separated tent weights $\mathbf{w}_i$ as shown in Figure 6.4.

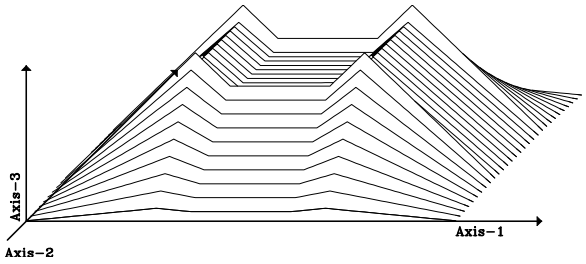

Figure 6.5: Superposition of overlapping patch weights. The shown quilt vector is used to balance the weighted patch contributions of the linear interpolation of neighboring, overlapping patches. The shown quilt vector is the result of zero padding and stacking of four slightly separated patch weights. The amplitudes indicate the number of weighted contributions each image quilt element received. The weighting operator $\mathbf{W}_m$ applies the inverses of these weight quilt elements to the corresponding image quilt elements. paper-quiltwt [ER]

Figure 6.6 demonstrates the effect of the weighted interpolation on the example in Figure 6.1. The top panel shows the input trace. The bottom panel shows an estimate of the signal's time variable variance. The weight $\mathbf{w}_m$, shown in the second panel, correct the brute stack, shown in the third panel, to the final result in the bottom panel.

IMPLEMENTATION AND INTERFACE

The programmer interface of the Patch classes successfully encapsulates and hides the somewhat complex implementation and invocation details of the underlying patch enumeration,

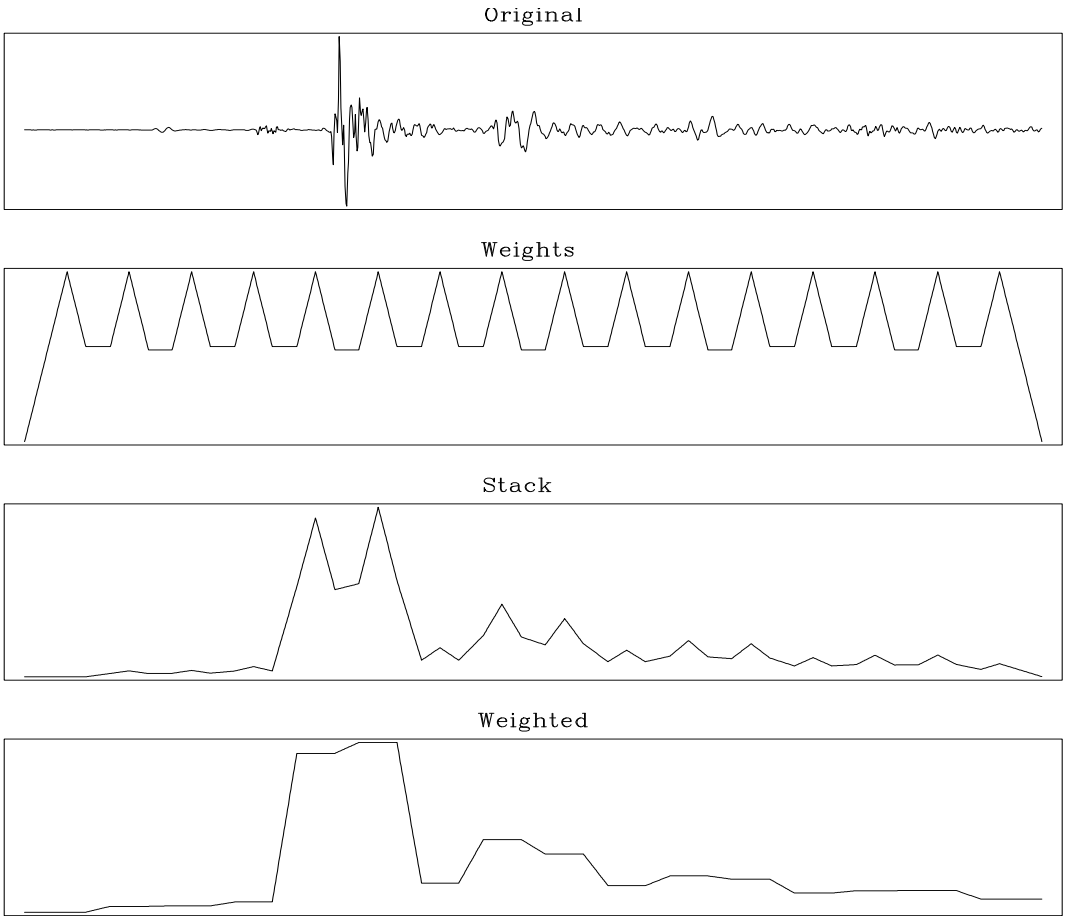

Figure 6.6: Variance computation of nonstationary seismogram. The top panel shows a seismic trace. The bottom panel shows the local variance of that trace. The two traces in the center panels display intermediate results of the algorithm. The quilt weight in the second panel is used to normalize the weighted stack in the third panel. paper-nStatVarEnum [ER]

windowing, weighting, and merging. The program `NStatVarPush` on this page shows the instantiation and invocation of the patch method. In general, The programmer chooses among two patch operators: PatchPush or PatchPull. PatchPush defines the patches as subsets of the input, while PatchPull defines them as subsets of the output. The initialization of the patch operators requires three arguments: the patch size and overlap, the input or output space, and an operator factory object. The patch operators implement Jest's operator interface (Schwab and Schroeder, 1997) and,consequently, the standard `image()` method accepts the input quilt and returns the processed output quilt.

```
package nstat.var;
import  jam.operator.*;
import  rsf.vector.*;
import  rsf.operator.*;
import  sep.io.ArgumentKeeper;
import nstat.patch.*;

public class NStatVarPush {
  public static void main(String[] args) {
    ArgumentKeeper argKeeper = new ArgumentKeeper(args);    // read args
    Rsf   iput = RsfFactory.newRsf    (System.in         ); // read input quilt
    float olap = argKeeper.getFloat   ("overlap"  , "0.5");
    int[] psze = argKeeper.getIntArray("patchsize", "321");

    OperatorFactory  opFac = new StatVarFactory();
    PatchPushOp      pchOp = new PatchPushOp((RsfSpace) iput.getSpace(),
                                             olap, psze, opFac);
    Rsf oput = (Rsf) pchOp.image(iput);                   // image quilt
    oput.write("pushOput.H"); iput.write("pushIput.H");   // write output quilt
  }
}
```

The patch operators require an operator factory to generate the stationary operators needed to process each stationary patch. Such a factory is usually easily programmed, because most of its methods are simply wrappers for the instantiation methods of the stationary operator.

Table 6.1: Some Operator Factory methods (syntax, effect).

`xspc = fac.createDomain(yspc)`	creates domain space for given range space
`yspc = fac.createRange( xspc)`	creates range space for given domain space
`oper = fac.getOp(xspc,yspc)`	creates operator given domain and range

Why are such wrappers necessary? A factory standardizes instantiation methods, which cannot be standardized directly by inheritance from an abstract interface (Gamma et al., 1994). Additionally, the operator factory supplies methods that return the domain given a range and vice versa. Table 6.1 shows the basic methods of the operator factory interface.

The programmer-friendly patch operators encapsulate the tedious and error-prone computations that split, process, and merge nonstationary data sets[1]. Internally, the patch operators invoke a class called PatchEnumeration that organizes the patch processing. The enumeration computes the actual patch location. The patch size is kept constant, but the overlap may slightly vary in order to fit the given patch size into the given input quilt size. The patch location is defined by the patch size and a patch anchor, the lowest-index vertex of the patch on the quilt space. The PatchEnumeration object directs a Window operator $\mathbf{P}_i$ to extract the various patches. For efficiency, the Window operator can change its domain and range without being instantiated. The operator handles minor rounding errors in the spaces' offset and increment intelligently, so that the patch's space remains a subspace of the quilt space. For each extracted patch, the patch operator requests a corresponding operator $\mathbf{F}_i$ from the operator factory. After processing the patch, the patch operator creates an interpolation operator $\mathbf{Q}_i$. The interpolation operator weights the output patch and adds it to the output quilt $\mathbf{m}$. The same interpolation operator images an identity vector and adds it to the weight quilt $\hat{\mathbf{w}}_m$. After the last patch is processed, the patch operator divides the output quilt's elements by the corresponding elements of the weight quilt. The encapsulation of the patch operator greatly owes its implementation to Jest's novel and careful formulation of domains, ranges, and vector spaces.

[1]Most programmers and readers do not want to know the details of the patch operator implementation and can skip this paragraph. But for anyone who wants to take a peek ...

EQUIVALENCE OF FILTER- AND DATA INTERPOLATION

To ensure smoothly space-variant filtering, we overlap neighboring patches, process each patch individually, and interpolate the resulting patch vectors. This patching operation may look ad hoc: Shouldn't we apply a spatially varying operator instead? In the case of a local-stationary filtering operation, the interpolation of filters and the interpolation of data patches approximately yield the same result, if the interpolation weights vary slowly with respect to the filter size. Under such circumstances, the sequence of convolution and interpolation can be commuted. Similar commutations should be permissible for other stationary operators.

I will demonstrate the equivalence of data and filter interpolation on a simple deconvolution example. The local-stationary data $z_0(\mathbf{x})$ has two stationary components. We find a normalized, smoothly varying, weighting function $w(\mathbf{x})$ that isolates the two stationary components. If we define the deconvolution filters for the two stationary components as $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, we can compute two filtered images

$$\begin{aligned} z_1(\mathbf{x}) &= f_1(\mathbf{x}) \star z_0(\mathbf{x}) \\ z_2(\mathbf{x}) &= f_2(\mathbf{x}) \star z_0(\mathbf{x}). \end{aligned} \tag{6.5}$$

The interpolation of the two filtered images yields the interpolation of the two deconvolved subsets:

$$\begin{aligned} z(\mathbf{x}) &= w(\mathbf{x}) z_1(\mathbf{x}) + (1 - w(\mathbf{x})) z_2(\mathbf{x}) \\ &= w(\mathbf{x})(f_1(\mathbf{x}) \star z_0(\mathbf{x})) + (1 - w(\mathbf{x}))(f_2(\mathbf{x}) \star z_0(\mathbf{x})). \end{aligned} \tag{6.6}$$

If, however, $w(\mathbf{x})$ is slowly varying compared to the size of the filters f_1 and f_2, the weight function can be assumed to be locally constant. Under this assumption, multiplication and convolution commute. Now, the filter – not the data – is interpolated:

$$\begin{aligned} z(\mathbf{x}) &= z_0(\mathbf{x}) \star \left[w(\mathbf{x}) f_1(\mathbf{x}) + (1 - w(\mathbf{x})) f_2(\mathbf{x}) \right] \\ &= z_0(\mathbf{x}) \star f(\mathbf{x}), \end{aligned} \tag{6.7}$$

where

$$f(\mathbf{x}) = w(\mathbf{x})\, f_1(\mathbf{x}) + (1 - w(\mathbf{x}))\, f_2(\mathbf{x}).$$

In terms of the two scales involved, the filter f varies smoothly on every scale, while the filter components f_1 and f_2 are locally constant.

DISCUSSION

Processing local-stationary data amounts to splitting, processing, and then merging stationary patches. The idea of patching is easily explained but its software implementation is tricky.

I redesigned Claerbout's original approach (Claerbout, 1992b) in an object-oriented framework. Due to Jest's handling of domain and range, the patch operators yield a high-level user interface that simply requires patch specifications and an operator factory object. The patches are specified by patch size and overlap in the domain *or* range. The operator factory generates a stationary operator given a specific domain and range patch.

Alternative approaches to process local-stationary data exist. In particular, I considered Claerbout's formulation of an underdetermined inverse problem that estimates an operator for each input sample. The discussion of filter versus data interpolation, discussed above, indicates that, for slowly varying local-stationary data, such a variable operator approach is equivalent to the chapter's patching scheme.

ACKNOWLEDGMENTS

Jon Claerbout's (1997) Fortran implementation provided the basic approach and test examples. Joel Schroeder implemented a preliminary version of the patching algorithm within the Jest framework.

Chapter 7

Algebraic Java classes for optimization

Jest (Java for estimations) separates optimization and application software without imposing limitations on either. Consequently, experts independently implement complementary and reusable optimization and application software, irregardless of the algorithms' internal complexity. The Jest library is accompanied by a growing number of examples and solvers. This chapter is an advertisement for the Jest software: my current version of Jest is available at my website[1].

Off-the-shelf optimization packages are usually limited to particular implementations and to somewhat arbitrary calling sequences. Off-the-shelf optimization packages usually implement their vectors in simple generic data structures, such as one-dimensional Fortran arrays. Unfortunately, generic data structures are often inconvenient or unacceptable in the case of complicated vectors or vectors too large to hold in-core. Alternatively, laboratories develop their own in-house optimization software that they adapt to their local data structure and application type. This approach, however, locks a laboratory into a fixed data structure and application style while simultaneously preventing any collaboration with laboratories of a different standard. Additionally, any new routine has to comply with the calling sequences of existing routines. A new routine can easily take advantage of old programs by invoking their

[1] `http://sepwww.stanford.edu/oldsep/matt/jest`.

known calling sequence. However, a new routine is not easily substituted for an existing alternative unless the two programs share identical calling sequences: a coincidence that is not easily maintained among a diverse group of contributing programmers.

Jest is centered around a set of abstract mathematical interface classes – *Jam* (Java for mathematics) – that resolve the implementation- and invocation-compatibility problem of traditional optimization libraries. Rather than defining a specific data representation, Jam defines the functionality of basic numerical analysis objects, such as vectors, vector spaces, operators, and solvers. Jam does not contain any implementations but consists of a collection of Java interfaces that assign names and methods to each of Jest's mathematical objects. The central Jam package is the only part of Jest that is shared by all users (see Table 71). All other Jest packages implement Jam interfaces. The individual The Javadoc pages available at our web page are

Jam interfaces do not prescribe any particular implementation, such as a vector representation. Collaboration among classes relies on the method invocations defined by the Jam interface. For example, Jam's vector interface requires all vectors to implement the simple Hilbert vector operations such as addition of two vectors, scaling of a vector, or the computation of the norm of a vector. A Jam-compatible class implements a Jam interface and limits itself to use only Jam-defined interface methods. a programmer of a Jest class is not concerned with the implementation or invocation of other Jest classes, but only with the invocation of the relevant Jam interfaces. Furthermore, the Java compiler ensures that any class that implements a Jam interface does indeed supply all necessary methods. Consequently, a programmer can, in principle, interchange all classes that implement the same Jam interface.

But how can one define a complete and general set of Jam interfaces? Mathematics! Jam defines names and method invocations for basic mathematical entities (vectors, operators, and solvers) and their standard operations. Ultimately, the Jam interface is as complete and general as the original mathematical definition. Furthermore, the mathematical framework of Jest is shared by researchers in numerical analysis, engineering, and natural sciences. A perfect interface design! However, only a full-blooded object-oriented language allows the necessary abstract formulation.

Package name	User Group	Class type	Vectors	Operators	Solvers
Jam	**Anyone**	Interfaces of abstract math objs	Vector	Operator CompoundOperator	Solver IterativeSolver
Juice	Mathematicians	Classes of numerical analysis	Rn	MatrixMultiplication	CG solver, Newton-Raphson ..
Jag	SEP geophysicists	Classes of seismic processing	Rsf, Isf	Nmo, convolution ...	
Wave	2nd research geophysicists	Classes of seismic processing		Kirchhoff migration ...	Arnoldi solver ...
Benzol	Chemists	...	...	...	...

Table 7.1: Package hierarchy. Jest packages (class libraries) are separated by user community. The top package, Jam, is shared by all users and only includes interfaces for abstract mathematical entities, such as a vector and its operations (add, scale, etc.).

Overall, Jam offers a plug-and-play quality: a programmer can accept or replace any part of Jest and preserve compatibility by implementing the corresponding Jam interfaces. Finally, Jam is not only general by not imposing a specific implementation, it is also extendible by allowing new mathematical object interfaces to be added.

Jest packages (class libraries) are separated by user community. The top package, Jam, is shared by all users and only includes interfaces for the abstract mathematical entities, such as a vector and its operations (add, scale, etc.). User groups (e.g., my laboratory's geophysicists) author packages the classes of which implement Jam interfaces. For example, Jag's Rsf class is my laboratories data structure and implements the Jam vector interface. The Juice package contains my elementary numerical analysis classes (e.g., my conjugate gradient solver). The bottom two packages do not exist but represent potential extensions by other user groups. Packages are bound to be compatible since they implement Jam interfaces. Consequently, any given user package only implements the classes that are unique to its user community. Jest's extendible structure and mathematical objects aims to garner collaboration across disciplines.

Jest's *Juice* package contains implementations of Jam interfaces: general solver classes (e.g., a conjugate-gradient solver), simple vector and space classes (e.g., a space of real numbers), simple operator classes (e.g., a zero and an identity operator), and classes of compound vectors and operators that wrap their components into a single vector respectively operator

(e.g., product vector or block operator). Since Juice classes are mostly trivial, light-weight implementations, we expect most Jest programmers to use them independent of their particular application.

Finally, the *Jag* (Java for geophysics) group of packages contains vectors and operators for seismic image processing. Jag is centered around the Rsf (Regularly Sampled Function) vector class. An Rsf vector represents a multi-dimensional regularly sampled, physical functions and is similar to a traditional float array type. An array of axis objects describes the Rsf vector space. Additionally, Jag comprises several seismic image processing packages based on Rsf vectors, e.g., convolution, edge detection, nonstationary patching, and missing data estimation. Within the Jest optimization framework, Jag's vector and operator implementations constitute a budding seismic processing library.

Java is a fun programming language. We struggled for years to develop an objected-oriented optimization and imaging package in C++, but our Fortran-programming colleagues found C++ too complex and difficult to master. We consider Java easy to program and hope, as Jest grows and prospers, some of our colleagues will join our effort.

We have little experience with off-the-shelf optimization packages since our laboratory, the Stanford Exploration Project, maintains its own small collection of optimization routines. It is this isolation of researchers and laboratories that Jest sets out to overcome. However, I will first list some brief and brash historic and technical comments on a few alternative libraries for contrast and comparison.

Jest is not the first library that attempts to separate solver and application software. Examples of well-known high-quality optimization libraries are MINOS (Murthag and Saunders, 1983 revised 1995) and MINPACK (Moré et al., 1980). However, these libraries are defined in procedural languages and impose severe restrictions on the application software. To the contrary, Jest does not require any particular implementation of any of its objects. It only requires that an object's mathematical operations are implemented somehow.

Historically, Jest is the latest in a series of optimization libraries. Our laboratory developed CLOP (1993), an experimental geophysical inversion library implemented in C++. CLOP enabled Lumley et al. (Lumley et al., 1994) to prototype a new multiple suppression scheme that

probably would not have been attempted in our laboratory's traditional Fortran environment. Nichols and Dye redesigned CLOP's class hierarchy to express abstract objects of vector algebra. Independently Gockenbach and Symes of the Rice Inversion Project developed a similar C++ library. In 1994, the efforts of the two laboratories merged into the Hilbert Class Library (HCL) (Schwab, 1994). Gockenbach published HCL (Gockenbach and Symes, 1996) and since maintains a detailed web-page (Gockenbach, 1996). Jest began as a Java implementation of HCL. We found Java much easier to use and learn than C++: an important fact for traditional Fortran or C programmers.

Lydia Deng et al. (Deng et al., 1996, 1995) developed a C++ optimization library for geophysical inversion that addresses similar problems as Jest. Fomel (Fomel and Claerbout, 1996; Fomel et al., 1996) focused on performance and implemented a geophysical optimization library in Fortran90 using its modular features.

In the summer of 1996 (before the development of Jest), SEP held an informal conference on objected-oriented numerics (Claerbout and Biondi, 1996). Afterwards, several participants cooperated to develop a framework to combine HCL's C++ classes with high-performance Fortran90 routines. The required linking of Fortran90 and C++ proved difficult but doable(Schroeder and Schwab, 1996; Urdaneta and Karrenbach, 1996).

Fall 1997, Jacob and Karrenbach (1997) implemented a multi-threaded velocity estimation based on Jest.

A DECONVOLUTION EXAMPLE

To introduce Jest we will discuss what probably is the *Hello World* program of image processing: image restoration by deconvolution. Many physical transmission systems blur their input signal. For example, atmospheric turbulences blur satellite and telescope images. The broadband wavelet in a seismic experiment blurs the reflectivity series of the Earth. Instruments themselves cause a degradation of their input which often is well described by a blur.

Mathematically, a blurred image $y(t)$ is usually modeled as the convolution of the original

image $f(t)$ by a blurring filter $b(t)$:

$$y(t) = b(t) * x(t) = \int b(t-t')x(t')dt'.$$

If the image is digitized, convolution can be expressed discretely as

$$y_i = \sum_j b_{i-j} x_j.$$

The discrete version can be formulated as a matrix multiplication $\mathbf{y} = \mathbf{B}\,\mathbf{x}$:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} b_1 & 0 & 0 & 0 & 0 \\ b_2 & b_1 & 0 & 0 & 0 \\ b_3 & b_2 & b_1 & 0 & 0 \\ 0 & b_3 & b_2 & b_1 & 0 \\ 0 & 0 & b_3 & b_2 & b_1 \\ 0 & 0 & 0 & b_3 & b_2 \\ 0 & 0 & 0 & 0 & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

which in this case implements transient boundary conditions.

In our example, we deconvolve the time series **y** recorded by a seismograph with known blur filter **b**. To estimate the unblurred image **x**, we seek the unconstrained minimum of

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{B}\,\hat{\mathbf{x}}\|$$

which yields the least square solution

$$\hat{\mathbf{x}} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{y}.$$

The routine that deconvolves the time series combines Jest vectors that represent **y**, **b** and $\hat{\mathbf{x}}$, a Jest linear-operator-with-adjoint to implement the convolution **B**, and finally Jest's conjugate-gradient solver to estimate the solution vector $\hat{\mathbf{x}}$. Figure 7.1 shows the known filter, the blurred input time series, and the deconvolved result.

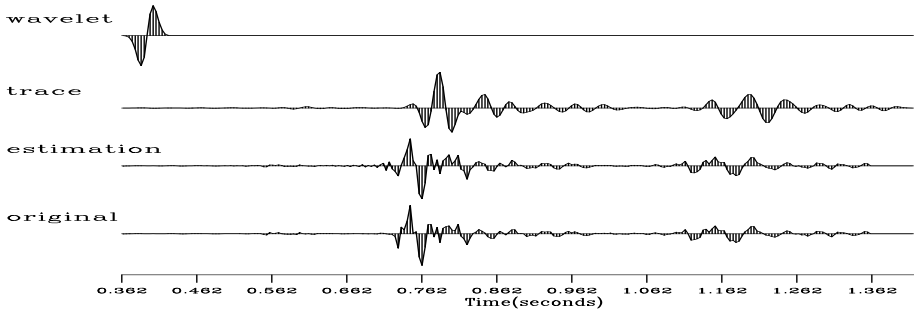

Figure 7.1: Seismogram deconvolution. The top signal is the known filter. The second signal is the received, blurred signal. The third signal is the deblurred signal estimated from the signals above. The bottom signal is the correct answer. paper-tcai [ER]

Table 7.2: Some Jest vector methods (name, syntax, effect).

add	`x.add(y)`	$x \leftarrow x + y$
scale	`x.scale(a)`	$x \leftarrow ax$
dot	`a = x.dot(y)`	$a \leftarrow \langle x, y \rangle$
getSpace	`x.getSpace()`	returns vector space

Vectors

The observed data **y**, the known blurring filter **b**, and the unknown deblurred image **x** are vectors. Each of them is accompanied by their own vector space. Vector and vector space are Java interfaces that define the basic vector operations, such as listed in Table 7.2 and Table 7.3. In the deconvolution example, the various vectors are represented by Rsf (Regularly Sampled Function) objects. The Rsf class is implemented as a multi-dimensional, in-core array. The RsfSpace class is an array of physical Axis descriptions. Rsf and RsfSpace are, of course,

Table 7.3: The Jest vector space methods (name, syntax, effect).

hasMember	`s.hasMember(x)`	true if vector x is member
newMember	`x = s.newMember()`	creates a new member x

Table 7.4: Some Jest operator methods (name, syntax, effect).

image	`y = f.image(x)`	$y \leftarrow f(x)$
image	`f.image(x,y)`	$y \leftarrow f(x)$
addImage	`f.addImage(x,y)`	$y \leftarrow y + f(x)$
getDomain	`f.getDomain()`	returns vector space

implementations of the general vector and space interface. and consequently implement the basic vector methods of Table 7.2 and Table 7.3. The Rsf and RsfSpace classes provide additional features, but for the moment it suffices to know that an Rsf contains a `write(filename)` method that stores a representation of itself and its vector space in a disk file. A class RsfFactory can read such a disk file and create the corresponding Rsf.

Operator

Theoretical considerations or problem descriptions often represent linear transformations by matrices. In scientific and engineering applications, as in the convolution example above, these matrices are often large and sparse. An implementation of such a transformation as an explicit matrix multiplication is inefficient. Instead Jest utilizes the more general concept of operator interfaces and leaves implementation details to each application. The operator concept, of course, includes the specific implementation of a linear operator as a matrix but also generalizes to nonlinear transformations. The convolution **B** in our example is a linear operator that implements Jest's operator interface. A few operator methods are listed in Table 7.4. The interface for a linear operator with adjoint is called *hasAdjoint* and extends the operator interface by a method `adjoint()` that returns the adjoint operator of the original operator. Many solvers require a linear operator's adjoint. The deconvolution example uses a simple convolution class `Tcai` and its adjoint; both implement the hasAdjoint interface. An auxiliary class TcaiFactory creates the needed transient convolution object.

Deconvolution

The code fragment `tcaiDecon` on the current page is the main routine of the deconvolution problem. The class instantiates two Rsf objects from corresponding data files. One contains the blurred time series, the other one the blurring filter. Next, the program instantiates a transient convolution operator, `convB`. Since the conjugate-gradient solver requires a positive-definite operator, we recast the example's operator and vectors according to the normal equations

$$\begin{aligned} \mathbf{B}\mathbf{x} &= \mathbf{y} \\ \mathbf{B}^T\mathbf{B}\mathbf{x} &= \mathbf{B}^T\mathbf{y}. \end{aligned}$$

The convolution's `convB.adjoint()` method yields the adjoint operator. Jest's CompoundLinearOperatorAdjoint class chains the forward and adjoint of `conv` to implement $\mathbf{B}^T\mathbf{B}$ in a single operator `convBn`. Jest provides a family of such compound operator classes. To create a starting vector for the solution, we request a new member from the domain of the operator. The values of the vector elements default to zero. Finally, we instantiate a simple CG solver that estimates the unblurred image. The integer argument at the solver's instantiation is the number of iterations and the arguments of the solve method are the elements of the normal equation: the compound operator $\mathbf{B}^T\mathbf{B}$, the known data vector $\mathbf{B}^T\mathbf{y}$, and the initial guess $\mathbf{x} = \mathbf{0}$ of the unknown model vector. The solver updates the vector $\mathbf{x}$ to its estimate of the model. The last line of the deconvolution routine writes the estimated image to the system's standard output stream. Figure 7.1 shows the inputs and the result of the inversion.

```
package conv.decon;
import jam.vector.*;
import juice.solver.*;
import juice.operator.*;
import rsf.vector.*;
import conv.convt.*;
/**
* I solve y = B x for x.                                    <BR>
* y is a    known time series                       (trace).<BR>
* B is a    known convolution operator of the blur (wavlt).<BR>
```

```
* x is an unknown relectivity.                              <BR>
* This creates the normal equations: y_n = B'y and B_n = B'B and
* calls a CGSolver to solve y_n = B_n x, respectively B'y = B'B x.
*/
public class TcaiDecon {
  public static void main(String[] args) {
    Rsf  yy = RsfFactory.newRsf("./trace.H");      // read   y
    Rsf  bb = RsfFactory.newRsf("./wavlt.H");      // read   b
    Tcai convB = new TcaiFactory(bb).              // create B
      getTcaiFromOutputSpace((RsfSpace) yy.getSpace());
    Vector     yn = convB.adjoint().image(yy);     // create yn = B'y
    CompoundLinOpAdj convBn =                      // create Bn = B'B
      new CompoundLinOpAdj(convB, convB.adjoint());
    Rsf xx = (Rsf) convBn.getDomain().newMember(); // create x
    new CGSimple(500).solve(convBn,yn,xx);         // solve  yn = Bn x
    xx.write( System.out );                        // write  x
  }
}
```

Solver

The conjugate-gradient solver that estimates the unblurred image vector $\hat{\mathbf{x}}$ implements the Jest solver interface. The solver interface defines a single method, `solve(operator B, vector y, vector x)`. The arguments stem from the equation $\mathbf{Bx} = \mathbf{y}$. The `solve()` invocation requires a starting solution for $\mathbf{x}$ which it then will update to its best estimate. The solver's termination criteria are usually set by the solver's constructor.

The solver program 7 literally translates the conjugate-gradient algorithm into Java code. Gill et al. (1981) list pseudo-code for the conjugate-gradient algorithm as:

$$p_k = -r_k + \beta_{k-1} p_{k-1} \tag{7.1}$$

$$\alpha_k = \frac{|r_k|_2^2}{p_k^T B p_k} \tag{7.2}$$

$$x_{k+1} = x_k + \alpha_k p_k \tag{7.3}$$

$$r_{k+1} = r_k + \alpha_k B p_k \tag{7.4}$$

$$\beta_k \quad = \quad \frac{|r_{k+1}|_2^2}{|r_k|_2^2}. \tag{7.5}$$

Each pseudo-code step corresponds to one or two lines of Jest instructions, as the program's comments indicate[2]. More importantly, the program's instructions are as general as the algorithm, since the program's objects – vectors and operators – are of the same abstraction as the algorithm's mathematical symbols.

```
package juice.solver;
import  jam.solver.*;
import  jam.vector.*;
import  jam.operator.*;
/** Conjugate Gradient Solver. */
public class CGSimple implements LinearSolver {
  private int maxIter;
  /** @param maxIter maximum number of iterations. */
  public CGSimple(int maxIter) { this.maxIter = maxIter; }
  /**
   * @param A operator to be inverted
   * @param b known vector
   * @param x starting estimate of unknown vector
   */
  public void solve(Operator A, Vector b, Vector x ) {
    if (! (A instanceof LinearOperator))
      System.err.println("Error: A is not a LinearOperator");
    Vector r = A.getRange().newMember(); // init r = b - Ax
    A.residual(x,b,r);
    r.neg();
    Vector p = (Vector) r.copy();        // init p = r
    Vector v = A.getRange ().newMember();// init v = 0
    float rtrNew = r.norm2();

    for (int iter=0; iter< maxIter; iter++) {
      A.image(p, v);                     //  v = Ap
      float ptv   = p.dot( v);           //ptv = p^t Ap
```

[2]Gill's algorithm and the implementation trivially differ in the representation of the operator by the letter **B** and the residual sign convention.

```
    float alpha = rtrNew/ptv;          //  a = |r|^2 / p^t Ap
    if (ptv == 0f) System.err.println("Error: ptv=0.");
    x.addScale(  alpha, p);            //  x = x + a p
    r.addScale( -alpha, v);            //  r = r - a Ap
    float rtrOld = rtrNew;
    rtrNew       = r.norm2();
    float beta = rtrNew/rtrOld;        //  b = |r|^2 / |r|^2
    if (rtrOld == 0f) System.err.println("Error: rtrOld=0.");
    p.addScale( beta, r, p);           //  p = -r + b p
}}}
```

CGSimple is a primitive member of Jest's solver family. Its entire interaction with a program is restricted to setting the number of iterations and invoking its solve routine. Standard iterative Jest solvers accept an iterator object that contains application specific initializations, termination criteria, report instructions, and final clean up code.

A MISSING DATA EXAMPLE

Jest is designed for complex, large-scale problems. To hint at the problems we had in mind when designing Jest, Figure 7.2 shows the result of a medium-sized missing data estimation (Claerbout, 1994). The estimation almost exclusively involves tools used in the deconvolution example above.

The panel on the left shows bathymetry (water depth) measurements $\mathbf{d}_k$ above the mid-ocean ridge in the Pacific. The measurements were taken by a side-scan sonar hanging off the bottom of a ship. The vessel's path did not cover the entire image area and consequently we attempt to estimate the missing data. The masking operator $\mathbf{M}_k$ extracts the known data, $\mathbf{M}_k\mathbf{d} = \mathbf{d}_k$ from the total data set $\mathbf{d}$.

The solution in the center panel minimizes the output of the Laplace operator applied to the data $\mathbf{d}$, $\min_{\mathbf{d}}(\nabla^2\mathbf{d})$, under the constraint that the known data does not change, $\mathbf{M}_k\mathbf{d} = \mathbf{d}_k$.

The solution in the right panel uses a two stage prediction-error technique. First, we minimize the convolution $\min_{\mathbf{p}}(\mathbf{p} * \mathbf{d}_k)$ to estimate the prediction-error filter $\mathbf{p}$. Next, we replace

Figure 7.2: Estimation of missing bathymetry data. The image in the left panel is the bathymetry data collected by a side-scan sonar along a vessel's path across the Pacific mid-ocean ridge. The image in the center panel shows an estimation of the missing data based on the convolution with a Laplacian. The image in the right panel shows an estimation by a two-stage linear prediction-error filter technique. paper-seabeam [ER]

the Laplacian in the earlier formulation by the prediction-error filter and solve again for the missing data. The missing data estimation is easily built from a combination of the objects shown in the deconvolution example, a specialized prediction-error filter class, and the mask operator $\mathbf{M}_k$.

DISCUSSION

Jest was designed to enable researchers to prototype complex inversion problems rapidly, to scale the prototyped solutions to large data examples, to maintain and reuse the software developed for different projects, to publish the results effectively, and to encourage collaboration among researchers of the same laboratory, of distant laboratories, and across disciplines.

Jest's advantages

Jest's high-level, mathematical encapsulation successfully separates numerical analysis into its solver classes, scientific application into its application classes, and data structure into its vector classes. Essentially, it is that natural separation that sets Jest apart from more common

optimization packages. The back-of-an-envelope designs of a researcher will usually translate directly to Jest's mathematical objects. Since the innovation will probably lie within a single area of expertise, such as a solver, or a new operator, a researcher will usually only need to program a single class. The new class can then be combined with Jest library items to build complex tests and examples. A small set of well-defined mathematical class interfaces ensures that such project-specific extensions are fully compatible with the entire Jest framework.

To help maintain Jest, its classes are accompanied by simple test routines. Furthermore, Jest includes a series of research-type examples that, due to our laboratory's reproducible research mechanism, are recomputed at the push of a button. To maintain Jest, we routinely and automatically execute all of Jest's test and example routines. Furthermore, the test and research cases complement Jest's Javadoc pages by offering a programmer illustrative examples.

In general, Jest's code is more readable than comparable C and Fortran implementations. On the numerical battlefield, when numbers are added and square roots are taken, Java is superior to C due to its lack of pointers and addresses. Here Java's readability is generally comparable to Fortran. On the abstract algorithmic level of solvers, Jest's mathematical objects are unsurpassed in conciseness and generality. Overall, as in every object-oriented language, a lot of *dirty* details, e.g., an array's parameters or input and output details, are hidden in Jest's objects.

For the future, I envision reproducible electronic documents that distribute scientific research on the World Wide Web. Reproducible research documents (Schwab and Claerbout, 1996; Schwab et al., 1996b; Schwab and Schroeder, 1995) organize an researcher's writings and his software implementation of his ideas into easily accessed and maintained packages. The online version of an electronic document, as shown in Figure 7.3, offers readers buttons that initiate the recomputation of the document's results.

Due to Java's security features, an electronic document implemented as a Java applet can be downloaded and safely executed by any standard Web browser. Thereby, Java enables us to combine a scientific article with its underlying software while avoiding the complications of traditional software installation and compilation. The resulting convenience and completeness

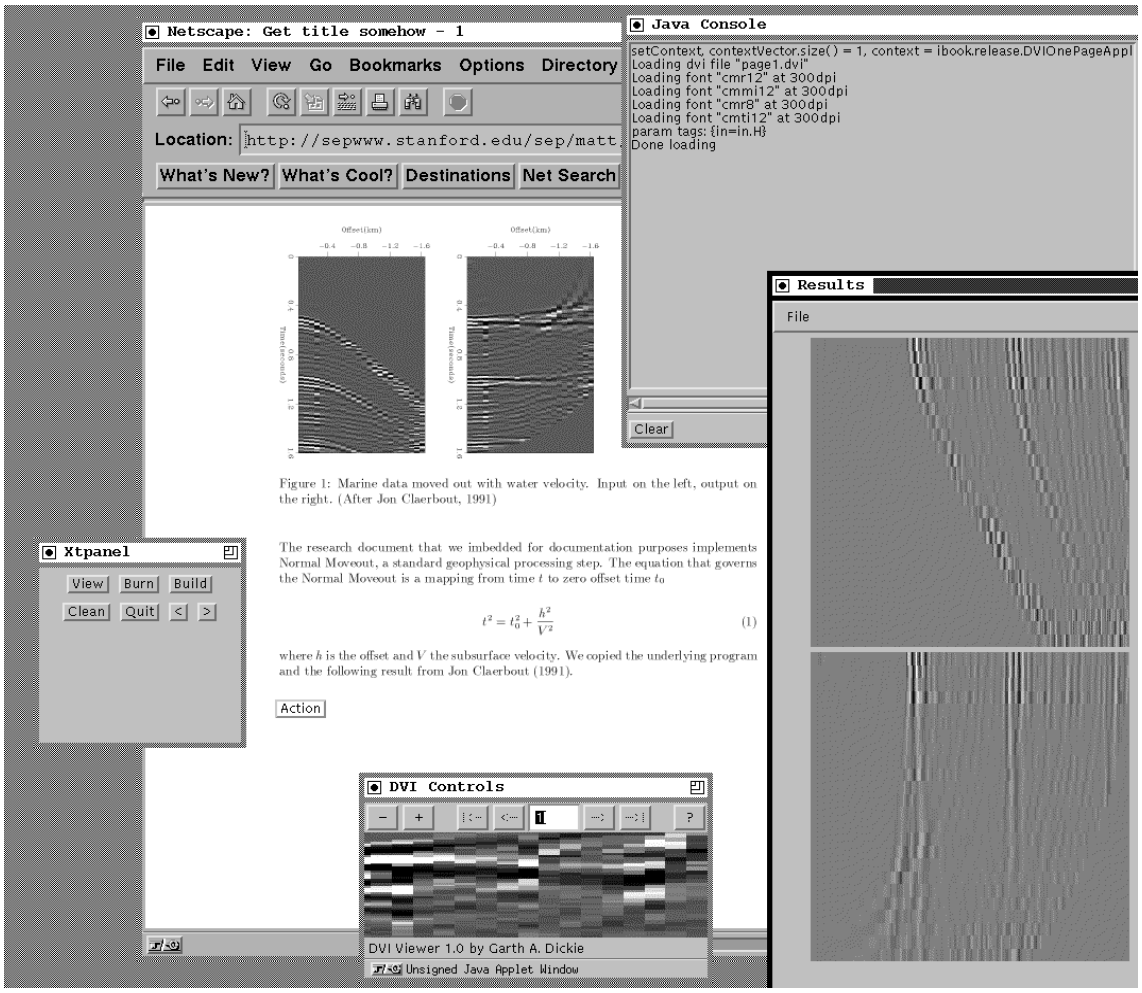

Figure 7.3: Reproducible electronic document on the Internet. A Java DVI viewer, IDVI (Dickie, 1997), displays a TEX research document in a Netscape browser. The window at the bottom center helps a reader to navigate the IDVI viewer. The Action button at the bottom of the white page activates a Java menu applet (left next to the button) that permits a reader to recompute the document's result. The browser downloads the necessary Java byte code from the server and executes them on the reader's machine. The Java console at the top right shows the executed commands. Another button enables the reader to download the document's corresponding application package. The final result figure is displayed in the window at the bottom right. paper-idvi [NR]

of reproducible Web documents should foster collaboration among researchers and increase research efficiency. Jest currently contains Java applications. However, first experiments (Figure 7.3) show how Jest's application can be extended to Java applets and how they can be integrated into a reproducible Web document.

... and disadvantages

Java is slower than C. When interpreted, Java is about 10 times slower. When compiled by a just-in-time compiler, Java is about 2 times slower. Experts predict that standard Java will soon be nearly as fast as C or C++, but probably never as fast as Fortran. Java's run time resolution of all methods that are not declared final and its array bound checks for indexing operations are probably the two features that hurt Jest's performance most. However, we have not yet investigated Jest's performance systematically.

Nevertheless, we intend to use Jest for large scale problems. To improve Jest's performance, we implement computationally intensive routines in C and link them with Jest objects. Java's native method mechanism easily facilitates the linking of C and Java routines. A Java program that invokes a C routine remains platform-independent if the method invocation is programmed to default to the Java version, when a native version is unavailable.

Typically, a Jest programmer prototypes and tests his program in an all-Java version. If the performance is insufficient, the programmer can profile the application and decide which part of the code to replace by a native method. We expect that straightforward numerical code, such as a convolution kernel, might need to be translated from Java to C, while algorithmically complex code, such as a solver, will not.

Alternatively, Jacob and Karrenbach (1997) report a performance comparable to Fortran when they combine Jest, a specialized compiler (JavaParty), and parallel computers in the optimization of a a large-scale geophysical inversion problem. Unfortunately, to yield that performance, their compiler required a non-standard class modifier regarding the distribution of parallelized objects.

Besides better performance, I wish Java contained a primitive type for complex numbers

and the ability to overload the array index operator `[]`. For the same reasons that Java offers float and double primitive types, it should include a complex-number primitive type[3]: (1) a complex-number primitive is important in many applications, (2) a class implementation incurs unacceptable inefficiencies, and (3) only a primitive type offers the concise and intuitive arithmetic operators, such as '+'. Alternatively, a complex number can be simply represented by two floats. However, the lack of a single complex-number type and its associated methods is error-prone, tedious, and not object-oriented.

For example, the methods below compute one root of the quadratic equation $ax^2 + bx + c = 0$. The concise first method does not exist in Java, since it uses a complex primitive type. The second method computes the same root but uses a user-defined complex class: It unfortunately obfuscates the simple, original mathematical operation.

Desired method:

```
static Complex root(Complex a, Complex b, Complex c) {
  return  (b - Complex.sqrt(b*b - 4*a*c)) / (2*a) ;
```

Available method:

```
static Complex root(Complex a, Complex b, Complex c) {
  Complex b2      = Complex.multiply(b,b);
  Complex ac4     = Complex.multiply(4,Complex.multiply(a,c));
  Complex discrim = Complex.sqrt(Complex.subtract(b2,ac4));
  return Complex.divide(Complex.subtract(discrim,b)),
                        Complex.multiply(2,a))); }
```

Without a primitive type for complex numbers, we must choose between an encapsulating complex-number class or an error-prone representation by a pair of floats. We partially sidestepped this difficulty by implementing a class for an array of complex numbers. Such an array justifies the class overhead, which was prohibitive for a single complex number, and simultaneously hides the representation of a complex number as a pair of floats. However,

[3]Fortran has a complex-number type, C and C++ do not.

using individual complex numbers remains inefficient or tedious unless Java adds a complex-number primitive type.

I wish the `[]` indexing operator could be overloaded in Java. Again, a multi-dimensional array is a fundamental structure in science and engineering applications. Internally the array might be stored as a Java array, a Java vector, or a huge out-of-core data file. Independent of the internal structure, applications need to index the array elements with a general indexing operation. Only the `[]` indexing operator offers the conciseness that matches the operation's simplicity. Since Java generally does not permit overloading operators[4], only classes with an internal Java array representation can offer the desired `[]` indexing operator.

For example, the program fragments 7 approximate the second derivative of a one-dimensional array. The first formulation with its natural index expressions is only available if the array is implemented as a native Java array. If the array is a user-defined array (e.g., an out-of-core, a complex number array) or a Java vector, Java's inability to overload the index operator leads to the verbose formulation of the second program fragment.

Desired method:

```
float x[] = new float[10];   // only native Java arrays
float y[] = new float[10];
...
y[i] =  (-x[i+1] + 2*x[i] - x[i-1]) / 4.0;
```

Available method:

```
Myarray x = new Myarray(10); // using Java's Vector
Myarray y = new Myarray(10); // or complex array class
...
y.setElement(i,(- x.ElementAt(i+1)
                + x.ElementAt(i  )
                - x.ElementAt(i-1)) / 4.0);
```

[4]Java makes an exception for the '+' operator in string concatenations.

In our applications, we could not forego the conciseness and simplicity of the array index expression and consequently had to pierce the encapsulation of our array-like vector classes to ensure access by index for operators. Therefore, the array-like Rsf vector and its operators are more closely linked than I wish. Additionally, our classes offer function calls to access individual array elements. However, I find such calls unintuitive, especially when the arguments include index and element values (`f.setDatum(3,3.f)`). I was not able to devise a general solution based on enumerations. In summary, I do not know how to write application classes with concise, but general, indexing operators for array-type data implemented by structures other than the original Java array.

I believe Java's future in science and engineering will be greatly enhanced if those two shortcomings - a primitive complex-number type and overloading the index operator - were redressed. Java designers should not underestimate the importance of science and engineering as sources of innovation and education.

Future

I continue to improve and enhance Jest by implementing my seismic imaging research in Jest. Jest needs more sophisticated numerical solvers. We may try to translate HCL's limited memory BFGS, implicitly-restarted Arnoldi, or augmented Lagrangian solver using a C++ to Java conversion script. Simultaneously, we seek collaboration with other scientists, engineers, and especially numerical analysists to share and develop the Jest framework. The current version of Jest is available at `http://sepwww.stanford.edu/oldsep/matt/jest`.

ACKNOWLEDGMENTS

Jon Claerbout suggested the separation of solvers and applications using object orientation. Dave Nichols guided me among half-a-dozen other programmers in the implementation of CLOP, a first implementation of Jon's idea. Independently, Dave Nichols and Lester Dye as well as Mark Gockenbach and Bill Symes developed a more rigorous framework that based its

class design on mathematical entities. I later helped Lester and Mark to unite the two frameworks into the Hilbert Class Library. Frustrated by HCL's inability to attract programmers among our laboratory's researchers, Joel Schroeder and I designed a Java library that rivals HCL but takes advantage of Java's simpler object-oriented features and its comprehensive library support. Thanks to all of you. It was great fun!

Chapter 8

Reproducible electronic documents

In the mid 1980's, researchers at our laboratory noticed that a few months after completing a project, the researchers were usually unable to reproduce their own computational work without considerable agony. In 1991, the concept of electronic documents solved this problem by making scientific computations reproducible. Since then Jon Claerbout, Martin Karrenbach, Joel Schroeder, and I developed electronic reproducible documents to our laboratory's principal means of technology transfer of scientific computational research. A small set of standard commands makes a document's results and their reproduction readily accessible to any reader. To implement reproducible computational research the author must use makefiles, adhere to a community's naming conventions, and reuse (include) the community's common building and cleaning rules. Since electronic reproducible documents are an easily maintained, reusable software portfolio, not only the reader but also the author benefits from reproducible documents.

REPRODUCIBLE RESEARCH

On average, two PhD students graduate each year from our laboratory, the *Stanford Exploration Project*. Years ago, junior students who built on their seniors' work often spent a considerable effort to merely reproduce their colleagues' old computational results.

Indeed, the problem occurs wherever traditional methods of scientific publication are used to describe computational research. In a traditional article the author merely outlines the relevant computations: the limitations of a paper medium prohibit a complete documentation including experimental data, parameter values, and the author's programs. Consequently, the reader has painfully to re-implement the author's work before verifying and utilizing it. Even if the reader receives the author's source files (a feasible assumption considering the recent progress in electronic publishing), the results can be recomputed only if the various programs are invoked exactly as in the original publication. The reader must spend valuable time merely rediscovering minutiae, which the author was unable to communicate conveniently.

To facilitate efficient technology transfer, our laboratory developed the concept of a *reproducible electronic document (ReDoc)*. A reader of a reproducible document can remove and rebuild the document's results without any application-specific knowledge. An author whose research involves scientific computations on a UNIX computer can easily create reproducible documents. Beyond a traditional article and the application's source code, a reproducible document contains three additional components: (1) makefiles, (2) a small set of universal `make` rules (less than 100 lines), and (3) naming conventions for files.

A makefile contains the commands to build a software package and is a standard UNIX utility for software maintenance. More powerful than a simple script of commands, a makefile has a notion that result files (*targets*) are up to date when they are younger than their corresponding source files (*dependencies*). The `make` utility, designed to maintaining software, elegantly maintains reproducible research projects. Fortunately, fine tutorial books about the `make` language exist(Stallman and McGrath, 1991; Oram and Talbott, 1991), and students easily begin using `make` even without formal introduction.

The second component in making scientific computations reproducible is the consistent availability of a small set of standard `make` rules. These rules allow a reader to interact with the document without knowing the underlying application-specific commands or files. The inclusion of an universal set of rules in every makefile ensures the rules' consistency across documents and enables authors to concentrate exclusively on the application-specific part of the makefile. Furthermore, a central set of rules accumulates the wisdom of a community on how to organize a reproducible electronic document. Our laboratory offers about 100 lines of

GNU `make` code (*ReDoc rules*) that, when included in the application makefile, implement a simple but powerful reader interface. In the electronic version of any reproducible electronic document, these ReDoc rules facilitate four commands: `make burn` removes the result files (usually figures), `make build` recomputes them, `make view` displays the figures, and `make clean` removes any files that are neither source nor result files. The process of recomputing the author's results allows a reader to understand and to modify the interaction of the various components.

The third component of a reproducible document system is its naming conventions for all files. These conventions allow a community to formulate universal rules that recognize a file's type and handle it appropriately. For example, a cleaning rule removes all intermediate files. The rule identifies intermediate files based on a community's naming conventions: typically files with suffix `.o` (object files) or files with the name stem `junk` (temporary files) are removed. A cleaning rule is important since a cleaned directory is more accessible and inviting to a reader than a cluttered uncleaned one. Furthermore, a cleaning rule saves resources such as disk memory by removing superfluous files. Naming conventions are also needed for result files. The rules for displaying, removing, or recomputing a result file are based on the result's file name. For example, at our laboratory result files (which are invariably figures) can have various formats such as postscript or gif. Our laboratory's naming conventions require the author to indicate the result file's format by a suffix, such as `.ps` or `.gif`. Consequently our laboratory can supply a universal format-independent rule for displaying result files: The rule identifies the result file's suffix, concludes the file's format, and invokes the appropriate viewing program such as `ghostview` for postscript or `xview` for gif files.

ReDoc rules are easy to use. An author who already uses makefiles only needs to adhere to the ReDoc naming conventions and include the ReDoc rules to make a traditional document reproducible. Our laboratory distributes its ReDoc rules, this article, and the accompanying example on its World Wide Web site(Schwab and Claerbout, 1996). A different community may need to adapt these ReDoc rules to its own peculiarities and naming conventions.

At our laboratory the software readily accessible to any researcher has increased tremendously. Today students commonly take up projects of former students, starting by easily removing and recomputing the original result files. Students who graduated and left our laboratory were able to seamlessly continue their own research at their new locations.

For example, in 1995, we successfully employed the ReDoc rules in our laboratory's sponsor report (14 articles by 15 authors) and three of Jon Claerbout's textbooks on seismic imaging. These documents contain a total of 483 result files: 276 easily reproducible, 21 conditionally reproducible, and 186 non-reproducible figures. Before publication automatic scripts removed and rebuilt all 276 easily reproducible result files (see Figure 8.1). We use the same scripts and documents to benchmark computer platforms. Additionally, our laboratory published 12 PhD theses that use an earlier version of the reader interface based on a dialect of `make` called `cake`. Before the web, we distributed these documents on CD-ROMs (Claerbout, 1996). Nowadays these documents are available on our web site, `sepwww.stanford.edu`.

We chose to implement the current reader interface in GNU `make`, since it is platform-independent, excels in the efficient maintenance of even complex software packages, and is equipped with a special mechanism to handle intermediate files, which we will discuss later (see section *Simultaneously clean and up to date*). Conceptually the ReDoc reader interface is independent of the document format (TEX, HTML, etc.) and independent of the underlying computational software, such as Matlab, Mathematica, or C and FORTRAN programs. Even though this paper restricts itself to UNIX systems and the `make` utility, the concept of a reader interface to reproduce a document's computational results should apply to electronic documents in other computer environments as well.

What is next? Of course, we want to publish our results on the World Wide Web. The Web conveniently distributes the combination of reading material for researchers and software for computers. Ideally, each computed figure in a future World Wide Web document should be accompanied by a *push-button* for the `burn, build, clean`, and `view` command. Currently we are closely watching the development of Java(SUN, 1996), a computer language for software on the Internet.

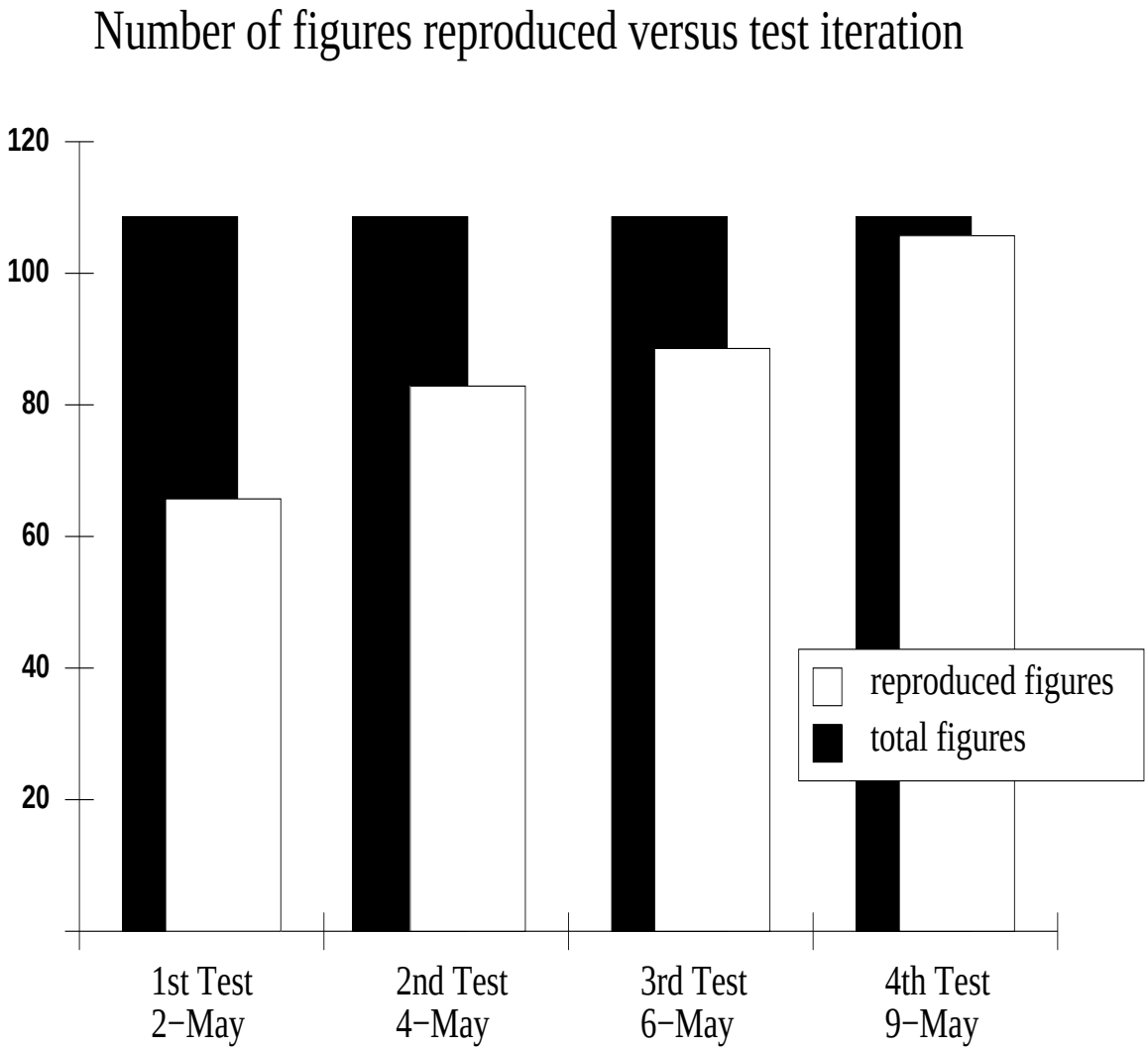

Figure 8.1: Quality control. A concrete test of a document's reproducibility is a cycle of burning and rebuilding its results. A simple script can implement such a reproducibility test by invoking the ReDoc rules described in this article. The ReDoc rules remove and regenerate the document's results independent of the document's content. The graph above plots the successfully reproduced figures versus the series of tests that removed and rebuilt the figures. The document contained 14 articles with 112 easily reproducible figures by 15 authors. After each test the authors were given time for corrections. After the first test, only 60% of the document's easily reproducible figures were in fact reproducible. After the fourth test, almost all figures were reproducible and the document was published. paper-col [NR]

BENEFITS TO THE READER

Just as a driver wants to find the brake pedal at the same location in every car, a reader of research documents wants a few standard commands to explore the scientific contents of any electronic document. The ReDoc rules offer four such standard commands:

- `make burn` removes the result files such as the postscript and gif version of a computed figure.
- `make build` recomputes the result files. It compiles the necessary programs, correctly invokes a sequence of executables, and finally stores the results in predefined files.
- `make view` updates the result files, if necessary, and displays them using the appropriate viewer, such as `ghostview` for a postscript figure, or the standard `cat` UNIX program for an ASCII file.
- `make clean` removes all the intermediate files that were created during the execution of the `build` command. It leaves the result and source files intact.

Consistent standard commands to remove and reproduce a document's result files not only help a reader access and study an unknown document, but also enable an author maintain his own software. A reproducible document is a research and software filing system. Authors document their scientific computations in the article and preserve the computational details in fully functional examples. Over the years, a researcher may assemble a portfolio of former research and software projects. The portfolio serves as reservoir of reusable programs and as exhibition of the author's abilities. The standardized commands of the ReDoc reader interface allow authors to easily test their archived research software by occasionally removing and regenerating the document's results. Furthermore, a community can develop automatic scripts to verify any document's completeness and reproducibility before its publication (see Figure 8.1). Publishers may envision that an electronic scientific journal could be refereed by testing the reproducibility of its illustrations.

Simultaneously clean and up to date

A document is best maintained both clean and up to date: each result file is younger than its ultimate source files, and the intermediate files are removed, but assumed to be up to date. A reader prefers a clean directory, since an uncluttered directory clearly presents the important source and result files. Simultaneously a reader expects reassurance that the document's results correspond with the existing source files. (Furthermore, we found that only cleaned documents are functional on a CD-ROM: intermediate files once stored on the read-only memory of a CD-ROM cannot be overwritten when the files are later regenerated by a reader.)

Unfortunately, popular `make` dialects generally do not support rules to keep documents simultaneously clean and up to date. These dialects consider a result file out of date when certain intermediate files are missing (for example, GNU `make` considers a result file out of date if a file is absent whose dependency is formulated by a non-pattern rule). Such treatment of intermediate files is convenient for software maintenance but not suitable for reproducible electronic documents.

The `cake` dialect of `make` was designed for document maintenance rather than for software maintenance. `cake` assumes *all* absent intermediate files to be up to date and therefore supports documents that are simultaneously clean and up to date. `cake` was originally introduced at our laboratory because it was the first freely distributed, platform-independent `make` dialect we found. Unfortunately, `cake`'s limited popularity at other sites made our reproducible electronic documents unattractive to potential readers.

Today's ReDoc rules are able to maintain a document clean and up to date while being formulated in the very popular GNU `make` dialect. At our request, Richard Stallman recently enhanced GNU `make` to adequately handle the ReDoc rules' intermediate targets.[1] He added a special built-in target, `.SECONDARY` that allows the author to choose the behavior of GNU `make` with respect to its missing intermediate files. If a makefile includes a `.SECONDARY` target without dependencies (the default at our laboratory), then every missing intermediate file is presumed up to date. The `.SECONDARY` target is implemented in GNU `make` versions higher than

[1] GNU `make` refers to intermediate files as *secondary files*. It uses the word *intermediate* in a slightly different, more restrictive sense.

3.74.

Degree of reproducibility

Since our laboratory deals with computational problems of various sizes using a diverse collection of software and hardware tools, not all result files are easily reproducible for every reader. Consequently, application makefiles typically define three result list variables: `RESULTSER`, `RESULTSCR`, and `RESULTSNR`. The endings `ER`, `CR`, and `NR` indicate to the reader the degree of reproducibility:

- **ER: Easily reproducible** result files can be regenerated within ten minutes on a standard workstation.

- **NR: Non-reproducible** result files, such as hand-drawn illustrations or scanned figures, cannot be recalculated by the reader.

- **CR: Conditionally reproducible** result files require proprietary data, licensed software, or more than 10 minutes for their recomputation. The author nevertheless supplies a complete set of source files and rules to ensure that readers can reproduce the results if they possess the necessary resources. Additionally the author lists these resources in a warning file (e.g. `myresult.warning`), which accompanies the conditionally reproducible result file (e.g., `myresult.ps`). In industrial-scale geophysical research, many interesting results are conditionally reproducible.

The standard make targets `burn` and `build` are complemented by targets `burnER, burnCR, burnNR, burnall` and `buildER, buildCR, buildNR, buildall`. For example `burnCR` burns all conditionally reproducible result files. The target `burn` is defaulted to `burnER` to restrict the standard removal of result files to easily reproducible ones. The target `build` is defaulted to `buildER` to recompute the result files that `make burn` removes.

EFFORT REQUIRED BY THE AUTHOR: AN EXAMPLE

The reader interface demands minimal effort from the author. At our laboratory,an author merely supplies the application-specific rules, while the community's ReDoc rules contain all definitions that are not application-specific. The author's definitions have to conform to the community's naming conventions, so that the author's application-specific rules can be invoked by the universal ReDoc rules.

The author of a reproducible document lists each result file as either easily, conditionally, or non-reproducible. For every easily or conditionally reproducible result, the author supplies a rule that generates the result file. Furthermore, the author specifies a cleaning rule that removes the intermediate files. The ReDoc rules offer the author a comprehensive default cleaning rule, `jclean`.

Since the author's rules deal with the document's application, the effort required of an author is best illustrated by an example. The electronic version of this article is accompanied by a subdirectory called *Frog*. Frog contains a complete albeit small reproducible electronic document about a finite-difference approximation of the 2-D surface waves caused by a frog hopping around a rectangular pond. Figure 8.2 shows the Frog example as a reader would find it on our CD-ROMs.

The actual electronic document comprises a directory with a few files. The short traditional scientific article that describes the implementation of the finite-difference approximation of the wave equation is available in three formats: LaTeX, PostScript, and HTML. Some RATFOR[2] files implement the 2-D wave propagation code. The `Fig` directory contains the result files: a figure (postscript and gif version) of the pond after some wild hops by the frog and the output (two float numbers) of a dot-product test of the linear finite-difference operator and its adjoint. To organize the document's files, the author of the Frog example wrote the following makefile:

```
SEPINC = ../Rules
include ${SEPINC}/Doc.defs.top
```

[2]RATFOR is a preprocessor for FORTRAN that provides control flow constructs similar to C. Many UNIX systems have the original AT&T RATFOR. Our laboratory distributes a freely available RATFOR on its World Wide Web server.

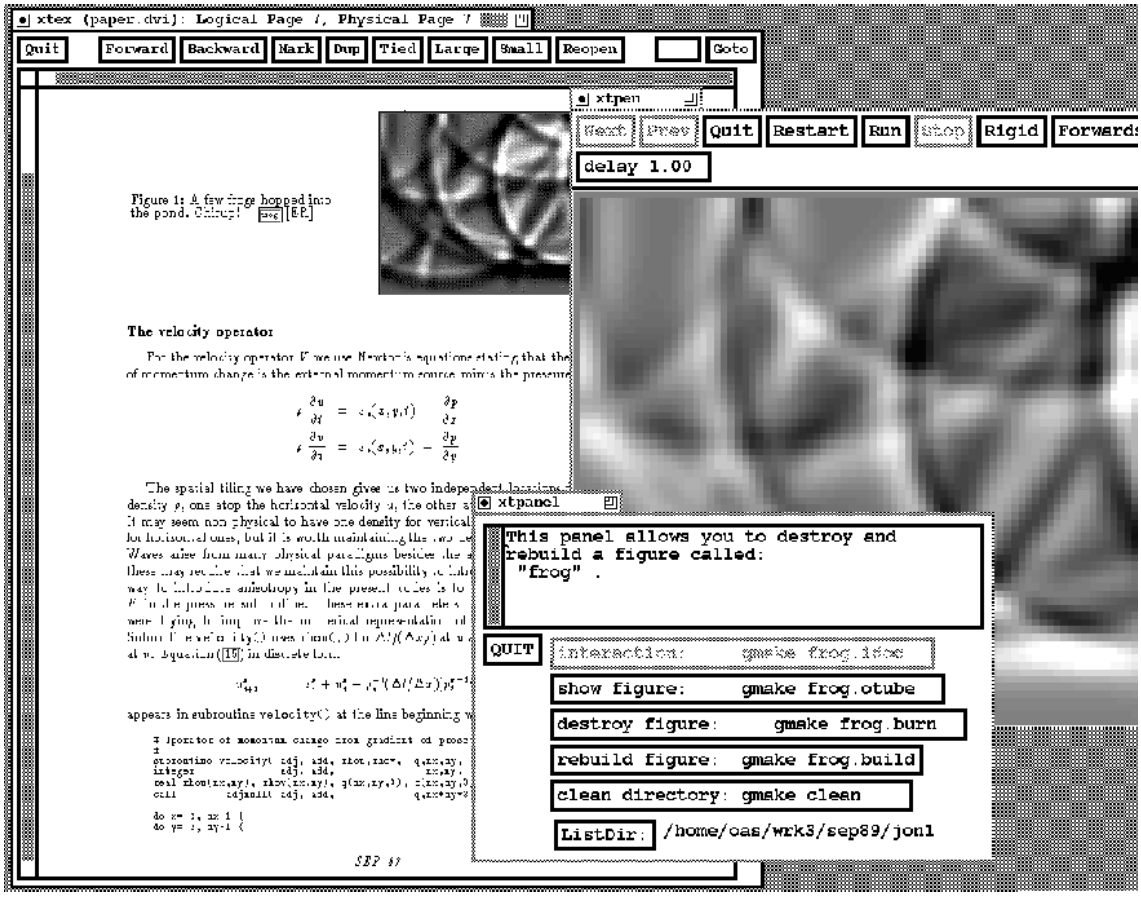

Figure 8.2: Online version of a reproducible electronic document. The reader interface for reproducible research is only one component of SEP's current computational research environment: A research document at SEP (visible in the background to the left) is written in LaTeX. Using SEP's own LaTeX macros, a push-button in each figure caption invokes a graphic user interface. The graphic user interface enables a reader to interactively execute the `burn`, `build`, `clean`, and `view` commands for each individual figure. (The panel is shown in the foreground. The result of `make view` is shown towards the right.) SEP's GNU `make` rules allow an author to extend the interactivity of a result figure easily to additional, application-specific actions. Unfortunately, these features are beyond the scope of this article. We distribute our software and the theses of our research group on World Wide Web. paper-xtpanelHor [NR]

```
RESULTSER = frog dot

col = 0.,0.,0.-1.,1.,1.
${RESDIR}/frog.ps ${RESDIR}/frog.gif: frog.x
        frog.x                  > junk.pgm
        pgmtoppm ${col} junk.pgm > junk.ppm
        ppmtogif        junk.ppm > ${RESDIR}/frog.gif
        pnmtops         junk.pgm > ${RESDIR}/frog.ps

objs =  copy.o adjnull.o rand01.o wavecat.o \
        pressure.o velocity.o viscosity.o wavesteps.o
frog.x: ${objs}

dot.build ${RESDIR}/dot.txt : dot.x
        dot.x dummy > ${RESDIR}/dot.txt
dot.view: ${RESDIR}/dot.txt
        cat ${RESDIR}/dot.txt
dot.burn:
        rm ${RESDIR}/dot.txt

dot.x : ${objs}

clean: jclean

include ${SEPINC}/Doc.rules.red
include ${SEPINC}/Doc.rules.idoc
include ${SEPINC}/Prg.rules.std
```

The variable `RESULTSER` contains the list of the document's easily reproducible results, `frog` and `dot`.

The next rule contains the commands to build the postscript and gif version of the `frog` result. Such a rule is application-specific and cannot be supplied by included default rules. The target names comprise the directory `RESDIR`, in which the result files reside, and file suffixes (`.ps, .gif`), which indicate the files' formats. The rule depends on an executable `frog.x`,

which it executes during the computations of the result.

Default rules for compilation and linking of executables such as `frog.x` are supplied by a shared include file, `Prg.rules.std`[3]. The dependency of the executable on its subroutine object files, as in the case of `frog.x`, needs to be defined by the author of the makefile, since it depends on the application-specific file names.

In the case of the `dot` result file, the rules supplied by the author reflect the commands of the reader interface: `dot.build` creates the result file, `dot.view` displays it, and `dot.burn` removes it.

Finally, the target `clean` invokes the included default target `jclean`. The targets remove intermediate files based on our laboratory's naming conventions.

OUR LABORATORY'S REDOC RULES

In the Frog example, a reader invokes targets, such as `build`, that are not listed in the author's application makefile. These targets are supplied by our laboratory's ReDoc rules and are merely included in the document's makefile (`Doc.defs.top, Doc.rules.red, Doc.rules.idoc`). They ensure a consistent reader interface, prevent the author from re-implementing the ReDoc rules in every makefile, and accumulate the wisdom of the entire community. They are formulated in a way that any individual author can override them. Experience shows, however, that overriding is hardly ever necessary or desirable.

An author or reader does not need to know the implementation details of the ReDoc rules to use the rules (most researchers at our laboratory have never inspected the ReDoc rules). But you may wonder how the rules operate and how you may have to adapt the rules for your community's computational environment.

[3]Compilation and link rules are compiler dependent. In the Frog example, we include some generic FORTRAN rules; at our laboratory compilation rules depend on an environment variable indicating the compiler type.

Burn

The `burn` rule invokes a chain of rules, which ultimately finds all easily reproducible result files and removes them. The `burn` target is included in every application makefile as part of the ReDoc rules.

```
burn: burnER

burnER: ${addsuffix .burn, ${RESULTSER}}
burnCR: ${addsuffix .burn, ${RESULTSCR}}

%.burn:
        ${foreach sfx, ${RES_SUFFIXES} ,\
          if ${EXIST} ${RESDIR}/$*${sfx} ; then \
             ${RM}    ${RESDIR}/$*${sfx} ;  fi; \
          }
```

The `burn` target invokes its dependency `burnER`. The `burnER` rule selects the easily reproducible result files for removal. The `burnER` rule uses GNU `make`'s built-in function `addsuffix` to generate its dependency list. Each entry of `burnER`'s dependency list is a concatenation of the name of an easily reproducible file and the suffix `.burn`. In the Frog example, `burnER` depends on `frog.burn` and `dot.burn`. The dependency `frog.burn` invokes the pattern rule `%.burn`, which removes the result files corresponding to the result `frog`. At our laboratory a single result name such as `frog` usually denotes several result files of identical contents but differing format, e.g. postscript or gif. The `%.burn` rule scans a list of possible suffixes (`RES_SUFFIXES = .ps .gif`) and removes all related result files: `frog.ps` and `frog.gif`.

Since text result files, such as `dot.txt`, are rare at our laboratory, the ReDoc rules do not contain laboratory-wide rules for handling them. Consequently the author of the Frog document supplies an explicit `dot.burn` rule in the makefile. This explicit `dot.burn` rule overrides the default `%.burn` pattern rule, which generates postscript result files.

The standard `burn` rule exclusively removes the easily reproducible result files. A reader can remove any existing, conditionally reproducible result files by invoking `make burnCR`.

A reader can exclusively remove the result files related to the `frog` result by invoking `make frog.burn`.

Build

The `build` rule updates, if necessary, the document's easily reproducible result files. The implementation of the `build` rule is similar to the implementation of the `burn` rule:

```
build:   buildER
buildER: ${addsuffix .build, ${RESULTSER}}
buildCR: ${addsuffix .build, ${RESULTSCR}}

%.build: ${RESDIR}/%.ps
```

At our laboratory, almost every result file is a figure that exists in postscript format. Consequently the ReDoc `%.build` rule updates the postscript version of any easily reproducible result, such as `${RESDIR}/frog.ps`. Additional versions of the result (e.g. `frog.gif`) are usually generated as a side effect of the rule that computes the postscript version (`frog.ps`).

As in the case of `dot.burn`, the nonstandard text result `dot` (since it is not a figure) requires the author to supply an explicit `dot.build` rule.

View

The `view` rule updates and displays the results:

```
view  : ${addsuffix  .view, ${RESULTSALL}}

%.view: FORCE
        if   ${CANDO_GIF}    ; then \
          ${MAKE} $*.viewgif ; \
        elif ${CANDO_PS}     ; then \
          ${MAKE} $*.viewps  ; \
        else \
```

```
  echo "can't make $*.viewps $*viewgif";\
fi
```

RESULTSALL lists all result files and is defined as the concatenation of RESULTSNR, RESULTSCR, and RESULTSER.

At our laboratory the %.view rule checks for the various formats of a result and chooses the first version the makefile knows how to generate. The variable CANDO contains the return value of a recursive gmake -n call. This return value indicates if that particular version of the result can be built. Having found a version that can be built, the %.view rule invokes another rule (%.viewgif or %.viewps) that updates and displays the result file:

```
%.viewgif  : ${RESDIR}/%.gif FORCE
        ${XVIEW} ${UXVIEWFLAGS} ${RESDIR}/$*.gif

%.viewps : ${RESDIR}/%.ps FORCE
        ${GVIEW} ${UGVIEWFLAGS} ${RESDIR}/$*.ps
```

In the Frog example, the frog.view rule finds a rule for computing a .gif version of frog. Consequently, it invokes the gif rule frog.viewgif. In return frog.viewgif executes xview to display the result file frog.gif. If your computer system does not support the gif viewer xview, then you will need to supplement the %.viewgif rule with your own display command. Alternatively, frog.viewps executes ghostview to display the result file frog.ps.

Clean

A community's cleaning rule is designed to remove the intermediate files and thereby to isolate the source and result files. A laboratory-wide cleaning rule attempts to recognize the intermediate files according to the community's naming conventions. Unfortunately, such a rule cannot possibly anticipate all names the author may choose for his intermediate files. Consequently, our laboratory does not supply a fixed, universal clean rule, but a jclean (*Jon's clean*) rule. jclean removes the files that adhere to our laboratory's naming convention for intermediate files. Every author is responsible to implement his own clean rule. Most authors at our laboratory accept the default cleaning rule by defining clean as:

```
clean: jclean
```

Some authors at our laboratory append the default `jclean` with a command to remove some additional files that do not adhere to the standard naming conventions. Only very few authors ignore the `jclean` target (and its communal wisdom) and design their own rule.

Since the author of the Frog example adheres strictly to the ReDoc naming conventions for files, the default `jclean` mechanism suffices to remove the intermediate files:

```
jclean : klean.usual klean.fort ;

KLEANUSUAL := core a.out paper.log *.o *.x *.H *.ps *.gif
klean.usual :
        @-${TOUCH} ${KLEANUSUAL} junk.quiet
        @-${RM}    ${KLEANUSUAL} junk.*

FORT_FILES = $(patsubst %.f,%,$(wildcard *.f)) junk.quiet
klean.fort:
        @\
        for name in ${FORT_FILES} ; do \
          if  ${EXIST} $${name}.r ; then  \
              ${TOUCH} $${name}.f ; \
              ${RM}    $${name}.f ; \
          fi ; \
        done
```

The `jclean` target uses two methods to identify intermediate files. The first method, `klean.usual`, simply removes files whose names fit one of the rule's name patterns: e.g. the executable `frog.x`, or the intermediate bitmap files `junk.pgm` and `junk.ppm`. The second method, `klean.fort`, removes FORTRAN files, such as `frog.f`, if RATFOR versions of the program, such as `frog.r`, exist.

Incidently, an alternative, more reliable cleaning mechanism would be possible, if we had direct access to the dependency tree of a given target. The anticipated cleaning mechanism analyzes the makefile's rules and dependencies to identify the intermediate files. Fastidious

authors would have the option of automatically removing all files that are neither source files nor result files. This alternative mechanism would free the author from naming the intermediate files according to the community's naming conventions.

DISCUSSION

Reproducible electronic documents offer standard commands to enable interaction with a document's results without learning any of the document's implementation details. Reproducible electronic documents invite readers to experience and exploit the author's work. The documents offer authors an automatic maintenance and quality control of their software. Creating reproducible electronic documents does add virtually no additional work to a research project that already uses the `make` utility to generate its results.

The GNU `make` rules that implement reproducible research have been very successful in organizing our laboratory's geophysical research. My fellow researchers use the rules daily. In the past three years, the laboratory has rigorously tested and published a dozen reproducible reports and theses: some on CD-ROM, some on the World Wide Web. Beyond the Stanford Exploration Project, Martin Karrenbach of Karlsruhe University and Bill Symes of Rice University adapted my set of GNU `make` rules for the needs of their research groups. David Donoho of Stanford's Statistics Department transfered the idea of reproducible research to his group's Matlab environment.

The dependency among files and the need to up-date them is of such fundamental organizational importance that I wish future operating systems and software libraries would directly incorporate these concepts. In particular, every file of an object-oriented system could offer two basic methods: a boolean query if the file is up-to-date and a command that instructs the file to up-date itself.

ACKNOWLEDGMENTS

Dave Nichols discovered `cake`, a predecessor of GNU `make`, and taught our laboratory how to use it. Jon Claerbout conceived the idea of reproducible research documents. He and Martin Karrenbach wrote `cake` rules for the four basic commands `burn`, `build`, `view`, and `clean`. Joel Schroeder and I redesigned the system from scratch and implemented in GNU `make`, a standard UNIX tool. We appreciate Richard Stallman's advice and his implementation of the special built-in target `.SECONDARY`. Bill Symes reviewed the ReDoc rules carefully. I thank him for his good-humor and suggestions.

Chapter 9

Conclusions

In this dissertation, I introduce and test traditional and innovative edge enhancement techniques to detect seismic image discontinuities. I did this research using a new object-oriented optimization software that I developed to tackle intricate and compute-intensive physical estimation problems. I implemented the software in Java, since the language promises portability, ease of programming, and potential execution within a World Wide Web browser.

To give a reader access to this publication's research, I organized this dissertation as a reproducible electronic document. The last chapter of this dissertation explains the universal concept of reproducible documents which a few colleagues and I developed during my years at Stanford University. Today's academic competitiveness and glut of information forces researchers to excel in the effective dissemination of their results. I believe the combination of the Internet, reproducible electronic documents, and Java research software, such as my optimization library Jest, can fundamentally improve the publication of computational research.

Overall, this dissertation has two aspects: the detection of seismic image discontinuities and the general software tools that I (with the help of some colleagues) developed to tackle this and similar research problems.

DISCONTINUITY ATTRIBUTES

Seismic volumes are difficult to interpret since a wide variety of geological features are mixed into a single complex image. Especially the detection of faults and other discontinuities is a difficult but essential step in the geological interpretation of seismic image volumes. A discontinuity attribute map attempts usually offers an easily interpreted alternative view of the original image by suppressing ordinary sedimentary bedding and enhancing the boundaries of sedimentary packages.

Traditional edge enhancement techniques locate edges by amplifying image components of rapid local amplitude change. Unfortunately, seismic images show the rapid local amplitude changes within sedimentary packages as well as at their boundaries. As demonstrated in this thesis, traditional edge enhancement techniques consequently fail to enhance seismic image discontinuities.

In contrast, the correlation coefficient of an image region and its best-fitting plane wave successfully suppresses sedimentary layers and delineates seismic image discontinuities. Furthermore, the technique reliably handles the sharp and smooth discontinuities.

Initially, I had hoped prediction-error filtering would excel at revealing seismic image discontinuities. Unfortunately, the quality of the discontinuity depends on the sharpness of the discontinuity in the original seismic image. Prediction-error maps delineate sharp image discontinuities, but unfortunately remove smooth ones. Consequently, many fault images that are zones of rapid change, rather than pixel-to-pixel discontinuities, are not delineated. In the case of sharp discontinuities, prediction error generates detailed fault maps that are distinctively different from maps representing plane-wave misfit.

Scientific problems can be categorized into two groups: (1) questions that lack an adequate explanation, formulation, or theory and (2) questions that are well understood but which exceed our resources to handle. The general geological interpretation of seismic images by a machine is a problem that is not yet formulated. This dissertation's enhancement of seismic image discontinuity is only one feature among many that could be automatically extracted. Future automatic classification software might locate salt bodies within an image volume or

identify particular stratigraphic surfaces such as on-lap, off-lap and erosional boundaries. I believe such a future system requires a comprehensive segmentation and classification scheme of seismic image features by simple local statistics, such as histograms. Similar classification and segmentation problems are successfully solved in satellite image processing (Sabins, 1996).

In contrast, plane-wave estimation, which is required by this dissertation's plane-wave misfit attributes, is a problem of the second kind: it is easily formulated as a nonlinear problem (Symes, 1994) but costly to solve. Symes, Claerbout, and I independently discovered an elegant and efficient solution by reformulating the plane-wave estimation as a sequence of two simple linear problems.

JAVA OPTIMIZATION LIBRARY

Jest, my extendible optimization library implemented in Java, enables researchers to prototype complex inversion problems rapidly and to scale prototyped solutions to medium data examples. As Java's computational performance improves, Jest will be able to tackle larger problems.

Jest's high-level, object-oriented, mathematical classes – such as vectors, and operators – successfully encapsulate numerical solver routines and scientific applications. Additionally, Jest's objects and methods directly reflect the familiar abstract mathematical entities and operations of numerical analysis. Consequently, Jest is applicable to any Hilbert-space optimization problem. All results in this thesis were computed using the Jest framework. A researcher with a working knowledge of numerical analysis should find Jest easy to use.

Since Jest is implemented in Java, Jest applications can potentially be executed by Web browsers. Combined with reproducible electronic documents, Jest can deliver not only reports about computational research but the research itself.

REPRODUCIBILITY

In an increasingly complex and information filled world, a reader's ability to find, understand, and assimilate an author's publication are often the most important ingredients to an author's success. Reproducible electronic documents offer a simple interface to the software underlying a computational research project. A set of four standard commands enables any reader to remove and recompute the results of any reproducible electronic document.

The reproducible electronic document is the standard publication format at the Stanford Exploration Project (SEP). A group of about fifteen researchers routinely remove and rebuild their results using my small set of *GNU make* rules that implement the four standard commands of reproducible documents at our laboratory. SEP's software maintenance, quality assurance, software exchange, and software reuse have greatly improved since the introduction of electronic documents. I hope other laboratories, besides Symes's Rice Inversion Project and Donoho's Wavelet research group at Stanford, will follow SEP's example and ensure the reproducibility of their research. Until recently our laboratory distributed its reproducible electronic documents on CD-ROMs. Today, the same documents are placed on SEP's web page.

The World Wide Web distributes electronic documents. Beyond the traditional paper copy, a minimal and realistic goal for scientific publications of computational research is the distribution of reproducible electronic documents. A Java push-button in an HTML page could download and recompute an author's results (if packaged as an applet). Authors could effectively offer their readers the underlying source code for adaption and modification. The World Wide Web, currently a huge dusty shelf of dead research articles, would come alive.

Bibliography

Bahorich, M., and Farmer, S., 1995, The coherency cube: The Leading Edge, **14**, 1053–1058.

Bally, A. W., 1983, Seismic expressions of structural styles: A picture and work atlas: American Associatioin of Petroleum Geologists.

Bednar, J. B., 1998, Least squares dip and coherency attributes: The leading edge, **17**, 775–776.

Bleistein, N., 1984, Mathematical methods for wave phenomena: Academic Press.

Bracewell, R. N., 1995, Two-dimensional imaging: Prentice-Hall, Inc., New Jersey.

Brown, A. R., 1977, Memoir 42: Interpretation of three-dimensional seismic data: American Associatioin of Petroleum Geologists.

Buckheit, J., and Donoho, D., 1996, Wavelab and reproducible research: `http://playfair.Stanford.EDU/~wavelab`.

Castagna, J. P., and Backus, M. M., 1993, Practice of AVO analysis: Society of Exploration Geophysicists.

Castleman, K. R., 1996, Digital image processing: Prentice-Hall, Inc., New Jersey.

Claerbout, J., and Biondi, B., 1996, Geophysics in Object-Oriented Numerics (GOON): An informal conference: SEP–**93**, 241–252.

Claerbout, J. F., 1992a, Earth Soundings Analysis: Processing Versus Inversion: Blackwell Scientific Publications.

Claerbout, J. F., 1992b, Nonstationarity and conjugacy: Utilities for data patch work: SEP–**73**, 391–400.

Claerbout, J. F., 1994, Applications of Three-Dimensional Filtering: Stanford Exploration Project.

Claerbout, J. F., 1996, CDROMs of the Stanford Exploration Project: `http://sepwww.stanford.edu/office/sepcd.html`.

Claerbout, J. F., 1997, Geophysical exploration mapping: Environmental soundings image enhancement: Stanford Exploration Project.

Davis, L. S., 1975, A survey of edge detection techniques: CGIP, **4**, 248–270.

Deng, L., Gouveia, W., and Scales, J., 1995, The CWP object-oriented optimization library: The Leading Edge, **15**, 365–369.

Deng, L. ., Gouveia, W., and Scales, J., 1996, The CWP object-oriented optimization library: `http://timna.Mines.EDU/cwpcodes/coool`.

Dickie, G. A., 1997, About IDVI: `http://www.geom.umn.edu/java/idvi`.

Fomel, S., and Claerbout, J., 1996, Simple linear operators in Fortran 90: SEP–**93**, 317–328.

Fomel, S., Crawley, S., and Clapp, R., 1996, A generic NMO program: SEP–**93**, 305–316.

Gabor, D., 1946, Theory of communication: IEEE Proc., **93**, 429–441.

Gamma, E., Helm, R., Johnson, R., and Vissides, J., 1994, Design patterns: Addison-Wesley Pub. Co.

Gersztenkorn, A., and Marfurt, K., 1996, Eigenstructure based coherence computations: 66th Ann. Meeting: Expanded Abstracts, Society of Exploration Geophysicists, 328–331.

Gill, P. E., Murray, W., and Wright, M. H., 1981, Practical optimization: Academic Press.

Gockenbach, M., and Symes, W., 1996, The Hilbert Class Library: a library of abstract C++ classes for optimization and inversion: submitted for publication in Computers and Mathematics with Applications.

Gockenbach, M., 1996, The Hilbert Class Library: A library of abstract classes for C++ optimization and inversion: `http://www.trip.caam.rice.edu`.

Golub, G. H., and Van Loan, C. F., 1989, Matrix computations: The Johns Hopkins Press Ltd., London.

Gosling, J., Joy, B., and Steele, G., 1996, The Java language specification: Addison-Wesley.

Jacob, M., Philippsen, M., and Karrenbach, M., 1997, Large-scale parallel geophysical algorithms in Java: A feasibility study: `http://wwwipd.ira.uka.de/~jacob`.

Jain, A. K., 1989, Fundamentals of digital image processing: Prentice-Hall, Inc., New Jersey.

Leon-Garcia, A., 1994, Probability and random processes for electrical engineering: Addison-Wesley Publishing Company, Inc.

Lumley, D., Nichols, D., and Rekdal, T., 1994, Amplitude-preserved multiple suppression: SEP–**82**, 25–45.

Luo, Y., Higgs, W. G., and Kowalik, W. S., 1996, Edge detection and stratigraphic analysis using 3D seismic data: Proc. 66th Annual International Meeting of the Society of Exploration Geophysicists.

Marfurt, K. J., Kirlin, R. L., Farmer, S. L., and Bahorich, M. S., 1998, 3-D seismic attributes using a semblance-based coherency algorithm: Geophysics, **63**, no. 4, 1150–1165.

Moré, J., Garbow, B., and Hillstrom, K., 1980, User guide for MINPACK: Technical Report ANL-80-74, Argonne National Laboratory, Argonne, IL.

Murthag, B. A., and Saunders, M. A., 1983 (revised 1995), Minos 5.4 user's guide: Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, CA.

Neidell, N. S., and Taner, M. T., 1971, Semblance and other coherency measures for multichannel data: Geophysics, **36**, no. 3, 482–497.

Newell, A., and Simon, H. A., 1975, Computer science as empirical inquiry: Symbols and search *in* Ashenhurst, R. L., and Graham, S., Eds., Turing Award Lectures:: ACM Press.

Nichols, D., Urdaneta, H., Oh, H. I., Claerbout, J., Laane, L., Karrenbach, M., and Schwab, M., 1993, Programming geophysics in C++: SEP–**79**, 313–471.

North, F. K., 1985, Petroleum geology: Allen & Unwin Inc., Winchester, Mass. 01890, USA.

Oppenheim, A. V., and Schafer, R. W., 1989, Discrete-time signal processing: Prentice Hall, Englewood Cliffs, New Jersey.

Oram, A., and Talbott, S., 1991, Managing projects with make: O'Reilly & Associates, Inc.

Prewitt, J., 1970, Object enhancement and extraction *in* Lipkin, B., and Rosenfeld, A., Eds., Picture Processing and Psychopictorics:: Academic Press.

Roberts, L. G., 1965, Machine perception of three-dimensional solids *in* Tippett, J. T., Ed., Optical and Electro-Optical Information Processing:: MIT Press.

Ronen, S., Geoltrain, S., Hattori, M., Schultz, P., and Varvik, G., 1993, Mapping reservoir properties from well measurements guided by seismic attributes: Proc. OTC Meeting.

Sabins, F. F., 1996, Remote sensing: W. H. Freeman and Company.

Schroeder, J., and Schwab, M., 1996, HCL and regular data: SEP–**93**, 263–272.

Schwab, M., and Claerbout, J., 1995, The interpolation of a $3-D$ data set by a pair of $2-D$ filters: SEP–**84**, 271–278.

Schwab, M., and Claerbout, J. F., 1996, SEP's reproducible electronic documents: `http://sepwww.stanford.edu/redoc`.

Schwab, M., and Schroeder, J., 1995, Reproducible research documents using GNUmake: SEP–**89**, 217–226.

Schwab, M., and Schroeder, J. M., 1997, Jest, an optimization library: `http://sepwww.stanford.edu/sep/matt/jest`.

Schwab, M., Holden, C., and Claerbout, J., 1996a, Revealing geological discontinuities by local plane wave suppression: SEP–**92**, 29–47.

Schwab, M., Karrenbach, M., and Claerbout, J., 1996b, Making scientific computations reproducible: SEP–**92**, 327–342.

Schwab, M., 1994, Birth of a C++ project: SEP–**82**, 251–256.

Schwab, M., 1997, Pre-whitening and coherency filtering: SEP–**94**, 189–204.

Sonneland, L., Barkved, O., and Hagenes, O., 1990, Construction of reservoir maps from seismic classifier maps: 60th Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 241–244.

Sonneland, L., 1983, Computer aided interpretation of seismic data: 53rd Annual Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, Session:S20.4.

Stallman, R. M., and McGrath, R., 1991, GNU Make: Free Software Foundation.

SUN, 1996, Java: Programming for the internet: `http://java.sun.com`.

Symes, W., 1994, The plane wave detection problem: Inverse Problems, **10**, 1361–1391.

Taner, M. T., Koehler, F., and Sheriff, R. E., 1979, See erratum. Volume 44, Issue 11, Page 1896.Complex seismic trace analysis: Geophysics, **44**, no. 6, 1041–1063.

Tarantola, A., 1987, Inverse problem theory: Elsevier.

Urdaneta, H., and Karrenbach, M., 1996, An IGF90 tutorial: SEP–**93**, 273–292.

Vail, P. R., 1977, Memoir 26: AAPG.

Weglein, A. B., 1989, What can inverse scattering really do for you today?: Geophysical Inversion Workshop, Siam, Geophysical inversion: proceedings, 20–45.