Irregularly-spaced, non-stationary signals

Jon Claerbout

ABSTRACT

Of three methods to deal with nonstationary signals the most appealing is to interpolate filters from a coarse mesh. While operators conveniently carry to a regular mesh scattered data *values*, scattered data *signals* invite techniques more akin to matrix inversion.

INTRODUCTION

My new book, to be my final book, GIEE (Geophysical Image Estimation by Example)[1] scarcely touched two important areas of application: (1) Dealing with nonstationarity by interpolating filters from a coarse mesh, and (2) Carrying irregularly-spaced geophysical data *signals* as opposed to *values* to a regular mesh. Here I provide some background for these two areas. Ultimately, both problems should be handled simultaneously.

NONSTATIONARY OPERATORS

Nonstationary data are those with spectra changing in time or space. The most common form of nonstationarity is waves changing their direction with time and space. Nonstationary data usually calls for nonstationary operators. We need those to accelerate solutions, to fill in data gaps, and to transform residuals to whiteness (IID).

Time-variable 1-D filter

My first go at nonstationarity was a time-variable PEF. Unfortunately, at the present state of computer hardware, the method is not suitable for multidimensional data. This method did work well in one dimension. Figure 1 shows synthetic data with time variable deconvolution. (More details are in the document labeled "Unfinished" at my website.)

The method is simple. Every point on the signal has its own filter. Because each data point has a multi-point filter, the PEF-design regression is severely underdeter-

[1]Not yet available on the retail market. Available from the manufacturer at lulu.com.

mined; but a workable regularization is forcing filters to change slowly. I minimized the gradient with time of the filter coefficients.

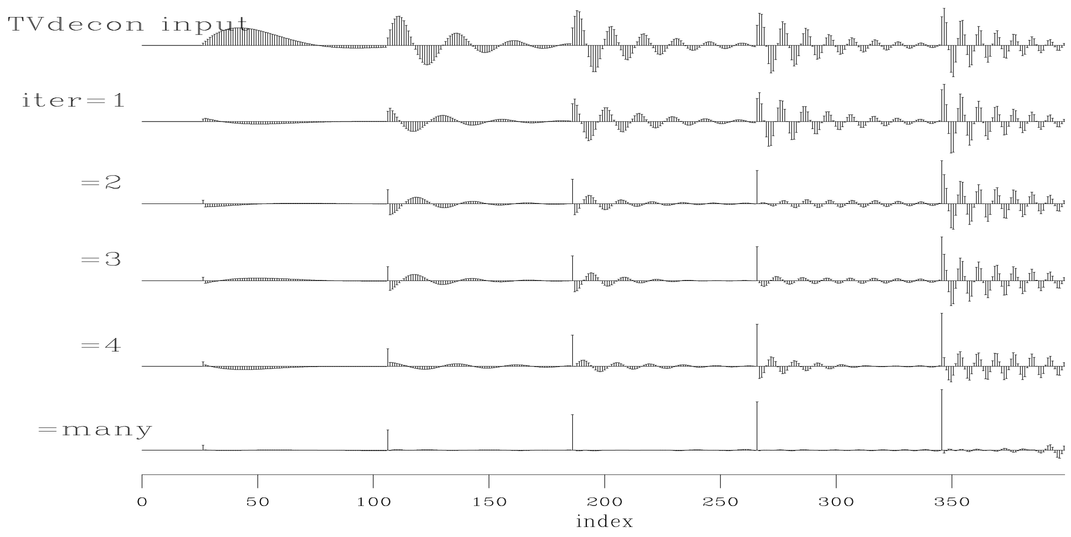

Figure 1: Time variable deconvolution with two free filter coefficients and a gap of 6. [**ER**]

As we hope for deconvolution, events in Figure 1 are soon compressed to impulses. The compression is remarkably good, even though each event has a different spectrum. What is especially pleasing is that satisfactory results are obtained after truly small numbers of iterations (roughly three). The example is for two free filter coefficients $(1, a_1, a_2)$ per output point.

Dip spectra commonly vary in time and space. In multidimensional spaces, we primarily struggle for machine memory. There, needing a filter array for each data point is abhorrent.

Patching

My second go at nonstationarity was patching. A big block of data is chopped into overlapping little blocks. The adjoint operation merges the little blocks back into a big block. The inverse patching operator is easily found by passing a big plane full of ones through the operator and back. What emerges will measure the overlap, i.e., find a bin count for a divisor to convert the adjoint to an inverse. Weighting functions of space may also be introduced and the inverse likewise calculated. Patching would appear to be well suited to modern parallel computer architectures.

Patches need not be equal in size nor be rectangular. Reflection seismologists immediately recognize the need for wedge-shaped patches in the space of time and source-receiver offset.

This method does work, but there are drawbacks. A big drawback is the many parameters required to specify patch sizes and overlaps. When PEFs are designed in blocks, then care must be taken to use internal filtering and attend to the fact that output lengths are shorter than input lengths. You live in fear that patch boundaries may be visible in your output. The many parameters increase the likelihood of miscommunication between the coder and the user. The many parameters also require effort and experience to tune.

Store the filter on a coarser mesh.

The first coarse-mesh filter idea is to keep the filter constant over a range of values in time and space. Such a filter would be easily stored on a coarser mesh, so the memory devoted to filters could be significantly less than the data. But, this idea evokes fear the outputs may show the blocky boundaries.

Bob Clapp (who has exercised nonstationary filtering in large-scale environments) suggests we should linearly interpolate filters from the coarser mesh. It can become costly, but economics are hard to figure in this age of rapidly changing computer architectures. Whether or not and how the coarse-mesh-filter idea is integrated with the helix transform is a topic that to my knowledge has not yet been attacked. The challenge for the analyst/coder is to produce filters interpolated from a grid in an environment that can be widely shared among many applications and with many people.

MOVING IRREGULARLY-SPACED SIGNALS TO A REGULAR MESH

Although we now have much experience taking data to a regular mesh bringing irregularly-spaced *scalars,* for irregularly-spaced *signals* we must do the problem at each time point, repeatedly solving it for each. There are, however, thousands of time points on a seismogram. Yikes! We need some way of accumulating and reusing knowledge. It seems we need something like an inverse matrix, but, that is exactly what GIEE has avoided, the reason being to avoid hopelessly large memory requirements.

We have accomplished much by using operators (function pairs) instead of matrices (data structures). We will soon see here that irregularly-spaced signals suggest a need for switching from operators to sparse matrices. Often, we move irregularly-spaced data to a regular mesh. We want the regular mesh for data viewing, filtering, correlation, and Fourier transformation. When we have data signals instead of simply data values, it seems we must repeatedly use the same iterative program at each point in time. But, sparse matrices might be vastly faster. To see why, represent a large collection of least-squares regressions (N time points), each with the same time-independent operator $\mathbf{F}^T\mathbf{F}$.

$$\mathbf{F}^T\,\mathbf{F}\,[\,\mathbf{m}_1\mathbf{m}_2\mathbf{m}_3\cdots\mathbf{m}_N] \quad = \quad \mathbf{F}^T\,[\mathbf{d}_1\mathbf{d}_2\mathbf{d}_3\cdots\mathbf{d}_N] \quad = \quad [\mathbf{b}_1\mathbf{b}_2\mathbf{b}_3\cdots\mathbf{b}_N] \tag{1}$$

Instead of iterating the same operator at each time, efficiency might be gained by approximating $(\mathbf{F}^T\mathbf{F})^{-1}$. How big is $\mathbf{F}$? We often deal with seismogram numbers from a hundred to a hundred thousand. Scalar data sets in GIEE range in size from a quarter to a half million. Model spaces there are typically larger because of zero padding. Examples there are solvable in a few minutes in desktop computers using operators. Industrial settings have comparable numbers of signals as GIEE has values. Luckily, industrial and some academic settings have clusters of computer cores, today numbering hundreds to tens of thousands.

Outside suggestions

I asked Michael Saunders for an approach using operators. He made two suggestions: First, scan the research literature, and Second, consider instead sparse matrices, in particular a technique known as sparse QR.[2]

Think of a sparse matrix $F_{i,j}$ as a list of three columns and K rows. Each row contains (matrix element, i value, j value). Observe how to multiply a sparse matrix times a vector, say $\mathbf{d} = \mathbf{F}\mathbf{m}$ or $d_i = \sum_j F_{i,j} m_j$.

```
do k=1,K
     data(i(k)) += matrix(k) * model(j(k))
```

A singular contribution of GIEE is multidimensional PEFs. Unlike gradient and Laplacian, PEFs are easily invertible, offering solution via a preconditioner $\mathbf{p}$ in which $\mathbf{m} = \mathbf{A}^{-1}\mathbf{p}$. In the simplest case $\mathbf{A}^{-1}$ would be leaky integration, trivially implemented with recursion. Recursion allows an easy solution to these huge problems. In the QR algorithm, recursion appears as the triangular matrix $\mathbf{R}$, in the name QR.

My preliminary ideas for how to do it

In practice the model mesh may always be dense enough that linear interpolation is adequate. We start from this assumption. As warm up, think about only one data signal in 2-D model space. On a first iteration, adjoint interpolation brings the data signal to its neighboring four mesh locations. A small number of iterations brings it to the surrounding neighborhood. When we need not fill a large region, not many iterations are required. In practice we push all data signals to the mesh at the same time. However, each time level requires us to solve an identical iterative problem.

[2]Michael Saunders recommends http://www.cise.ufl.edu/research/sparse/SPQR/ a sparse QR method and code by Tim Davis.

As there are typically thousands of time points, those iterations get tiresome! Let us solve this problem at each of about 40 time levels. Then let us see how we might use these results to more quickly obtain mesh values at the remaining thousands of time levels.

Limiting calculation to the 40 time levels, consider each mesh signal separately. Correlate the mesh signal with all the data signals. Select data signals with the strongest correlation. Using only those data signals, find the coefficients defining the best linear combination of data signals. Use these coefficients to define this mesh signal for all other times. The idea of using only data strongly correlated to the mesh signal could be made more sophisticated, and perhaps better. Limiting to 40 time points, using all data signals to fit the mesh signal we could jettison data signals from the fitting by applying some ℓ_1 penalty to the fitting coefficients.

Alternate view

I would be more satisfied with this algorithm if instead of 40 time levels, it dealt with 40 time lags. But I don't know how to put that together.

CONCLUSION

Well my friends, we have come a long way, and made much progress. I have grown old, so it may be up to you to flesh out the theory, write the code, and produce the examples, thereby uncovering the pitfalls. I'll try to interest some young person to take on these projects, but don't hold your breath.