# Industrial scale high-performance adaptive filtering with PEF applications

*Kaixi Ruan, Joseph Jennings, Ettore Biondi, Robert G. Clapp, Stewart A. Levin, and Jon Claerbout*

## ABSTRACT

In most areas of SEP research, neither data nor models are truly stationary. As a consequence, global fitting, regridding, and inversion rarely approach the ideal of independent, identically-distributed residuals. Here we design and develop parallel multidimensional adaptive filter routines parsimonious in both memory and computation, suitable for industrial scale applications. We applied them to examples of three classes of time and space variable adaptive Prediction Error Filters (PEF) applications: (1) blind decon, (2) filling gaps in data, and (3) making random realizations of data; and then, outline other areas where these tools will have a wide application.

## INTRODUCTION

Estimating data covariance, or in the case of PEFs the inverse covariance, is a powerful tool with a wide range of applications. Its uses include decon, missing data interpolation, signal and noise separation, and fault detection. Because both seismic data and the subsurface properties we want to infer from them are nonstationary, we need time- and space-variable PEFs.

Time-variable filtering has a long history with space-variable filtering only a few steps behind. In the context of very large, e.g., terabyte, datasets, big challenges must be overcome to be practical, as follows:

- Constraining memory usage
- Minimizing arithmetic complexity
- Simplifying the parameter space
- Enabling an efficient parallel implementation
- Avoiding instability
- Constructing a tool that can easily be shared in a wider context with many other people.

Over the last twenty-five years several attempts of capturing non-stationary covariance have been attempted at SEP. Schwab (1998) solved a series of stationary problems using overlapping patches. The downside of using patching is the large parameter space (size of patch, amount of overlap) that must be searched to find an acceptable result. Crawley (2000) solved for a global non-stationary covariance by creating micro-patches and requiring them to vary smoothly. The micro-patch approach adds significant memory requirements (for each data point you must store which micro-patch with which it is associated) and the resulting indirection reduces performance. A third approach is to linear interpolate from a sparse grid of filter locations. The downside of this approach is that it requires interpolating the filter at every location. The interpolation cost can be minimized by being careful on how you interpolate between the sparse grid of filters Hale (2006). Here we use this sparse grid of filters approach.

In this progress report we describe our method and show a small number of preliminary results on 2-D seismic data, 2-D images, and on 3-D synthetic data. We have not yet done enough tests to recognize all its opportunities and limitations. We have not yet documented a clean user interface. But we think the code is working and look forward to providing more examples and documentation for general use.

First we summarize the basic principles of multi-dimensional prediction-error filtering (PEF). Then we describe our Fortran 2003 implementation of non-stationary filter estimation. Finally, we show examples that support our claim to have built an important basic tool. However, we have not yet tested use of this tool in general inversion problems other than deconvolution and interpolation.

## PREDICTION ERROR FILTERS

Before diving into the enticing details, recall the textbook definition of a multidimensional prediction error filter (PEF) which underlies our time-variable IID (TV-IID) applications.

In one dimension, it vanishes before $t = 0$; it takes a unit value at $t = 0$; and it has coefficients determined by least squares for $t > 0$. In 2-D, the unit value is along one side of a rectangle. Analogous to the 1-D PEF vanishing before $t = 0$, in 2-D there are zero values along one edge of the rectangle before (or after) the unit value. In 3-D the unit value is along one face of a box, as depicted in Figure 1. The enticing mathematical feature of PEFs is that their filter outputs are white. More precisely, the output tends to whiteness as the filter size increases. This fact is shown in GIEE Claerbout (2014) chapter 7 (based on chapter 4).[1]

In multidimensional PEFs, once their coefficients have been learned (by least squares), contain the inverse spectrum of the input data. The PEFs are valuable

[1]While that source only proves whiteness in a 1-D sense, its helix interpretation of the convolution of multidimensional data and filter as an equivalent 1-D convolution of unwrapped data and unwrapped filter imply the $N$-D result.

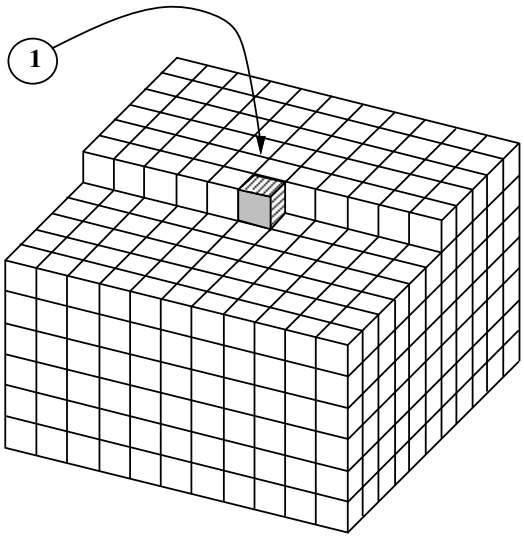



Figure 1: A 3-D causal filter at the starting end of a 3-D helix. [**NR**]

because they have many uses that include:

1. Conditioning residuals for inverse problems
2. Blind deconvolution
3. Creating random realizations
4. Filling missing data

Figure 2 shows the boundary effects that arise when the PEF is not allowed to access data outside a rectangular mesh. (It is left-right reversed compared with Figures 4-6.) Zeros around three edges of our Figures show the filter size.

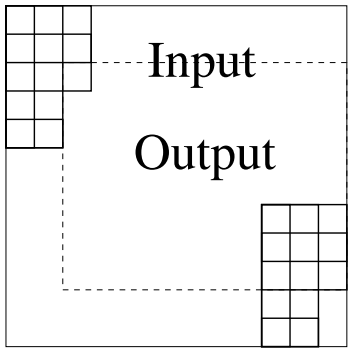



Figure 2: Domain of inputs and outputs of a two-dimensional filter like a PEF. [**NR**]

## DESIGN CONSIDERATIONS: THREE BOXES

Confusion may arise because distance scales arise in three contexts, (1) data fabric, (2) filter, and (3) blending. In 2-D these distance scales define boxes. In 3-D cubes.

1. Fabric box. Data being investigated will have characteristic lengths relating to how fast its spectrum changes. This box subjectively defines the size of a region where the spectrum is roughly constant. For example, you might presume on one side of a CMP gather that hyperbolic moveout roughly allows interpolation along 3 to 4 straight lines.

2. Filter box. The application programmer or end user must choose the filter size. It is given in points on the axis, e.g., (20, 2) might mean a filter 10 points long on the time axis acting on 2 neighboring traces.

3. Blending box. The spacing on the coarse mesh that filters are defined. They are linearly interpolated to the data mesh. For example, a blending box size of (1/3, 1/3) means we count 3 boxes along each axis meaning 4 points along each axis at which filters are defined.

Having a smaller blending box than the fabric box allows the filter field to adapt to the local degree of nonstationarity. On the other hand, too small a blending box introduces the danger of overfitting. Examples in our immediate future should provide us a better understanding of appropriate ratios among these three box sizes. Unfortunately, the fabric box size is intrinsically space variable so ultimately we appear to be facing deeper issues sometimes glibly called "deep learning." Hopefully, by limiting our examples to realistic seismic data we can provide users with useful guidance for parameter setting.

# METHOD AND IMPLEMENTATION

Our TV-IID operator applies a linear interpolation to filter coefficients defined on a coarse grid to calculate filter coefficients for each point in the denser data mesh. These interpolated filter coefficients are then applied at that data mesh location. Although it is theoretically equivalent to the chain of two well-known operators that we already have in the current library, we did not adopt this approach because it is memory intensive. We concoct a new operator that blends the two in a computationally efficient way to make it practical.

## Filter design

We designed our linearly varying filter in an object-oriented manner using Fortran 2003. The main filter class contains a 2-D `lag` array that contains the coordinates describing the shape of the filter and also two arrays that contain locations of the filter coefficients on the coarse grid. The class requires that the user provide the filter spacing they desire on each axis from which it computes and builds bounds for the blending boxes `b`{egin} and `e`{nd} arrays.

This filter type also contains an array containing the estimated filter coefficients. We have designed this array for the first dimension to contain the filter coefficients, and the remaining dimensions to be the number of filters along each axis present in the data. With this design, the number of dimensions of this array will grow with the size of the problem. To account for this growth, we extend our filter class, to 1-D, 2-D, 3-D, ... types.

## Code parallelization

Because the filter coefficients on a data point only depend on the patch inside which it resides (they are interpolated from those defined on the corners of the patch), each patch can perform the TV-IID operation independently. In our code the outermost loop is over patches and we can easily use tools such as OpenMP for parallelization.

## Fast linear interpolation in one dimension

Given an $n$ points 1-D grid and any filter coefficient `f_1` at the beginning and `f_n` at the end of the grid, the traditional linear interpolation looks like the following:

```
real :: g,h
integer :: i,n
do i=2,n-1
   g=(i-1)/(n-1)
   h=1-g
   f_i= h*f_1+g*f_n
end do
```

Fast interpolation removes a division, subtraction and two multiplications from the inner loop, as follows:

```
real :: delta
integer :: i,n
delta=(f_n-f_1)/(n-1)
f_i = f_1
do i=2,n-1
   f_i= f_i+delta
end do
```

We can trace the history of this algebraic reorganization back to Newton's method of divided differences (Newton (1687); Stirling (1730)). It is the simplest example of interpolation by numerical solution of differential equations.

## Design discussion

We deliberately did not use an explicit memory array for the values intermediate between `f_1` and `f_n` because they can, and usually would, be computed and applied

on the fly.

Our "fast interpolation" is *recursive* in the sense that filter coefficient updates are tied to local shifts of the filter by 1 grid point. It is not a significant impediment for this method because: (a) Our main parallelism is over an outer loop of patches, (b) If we start from the center of a patch, we can simultaneously work both left and right, up and down, etc., i.e., $2n$ directions in $n$ dimensions[2], and (c) Even these inner loops could be unrolled to compute interleaved entries with a multiple of `delta` should the need arise.

While the difference in arithmetic cost is an attractive factor of approximately three per dimension, an equally important benefit in higher dimensions is that we avoid repeatedly accessing coarse grid filter coefficient arrays (`f_1` and `f_n` in the above example) that are widely separated in memory.

## Code samples

Using the above example, the 1-D forward operator is shown here. The reader may easily identify the part of fast linear interpolation and the part of convolution.

```
subroutine forward1D(m,f,d) !output d=f*m
      real, dimension(:) :: m
      type(lv_filter1d)  :: f
      real, dimension(:) :: d
      integer            :: i,j,id
      real,allocatable   :: b0(:),d0(:)

      allocate (b0(size(f%coef1,1)),d0(size(f%coef1,1)))

      do i = 1,size(f%b,2)         ! Loop over patches
        b0=f%coef1(:,i)
        d0=(f%coef1(:,i+1)-b0)/(f%e(1,i)-f%b(1,i)+1)
        do id=f%b(1,i),f%e(1,i)    ! Loop over each point in the patch
          do j = 1,size(f%coef1,1) ! Loop over the filter coefficients
            d(id+f%lag(1,j))=d(id+f%lag(1,j))+b0(j)*m(id)
          end do
          b0=b0+d0
        end do
      end do
      deallocate(b0,d0)
end subroutine forward1D
```

The above example implements the concept of convolving a filter and vector to produce another vector. We can write this in matrix form in two different ways. The first approach is

$$\mathbf{d} = \mathbf{M}\mathbf{f}, \tag{1}$$

[2]Snaking back and forth columns and rows helps preserve locality and reduce cache misses. More sophisticated "space-filling paths" (see, e.g., Savage (1997); Knuth (2005); Flahive (2008)) that tend to optimize parallel locality are also possible.

where $\mathbf{M}$ is convolving with our model, $\mathbf{f}$ is our filter, and $\mathbf{d}$ is our data. The second approach is

$$\mathbf{d} = \mathbf{F}\mathbf{m}, \tag{2}$$

where $\mathbf{F}$ is convolving with our filter, and $\mathbf{m}$ is our model. The two approaches imply two different adjoints. The first approach, equation 1 is needed to estimate the filter. The second approach, equation 2 is needed to apply the inverse covariance to a model. We have implemented both types of adjoints for up to three dimensional models.

# EXAMPLES

We tested our library on several different datasets with different objectives. In our first example we looked to see how our non-stationary filter performed when confronted with an image with sharp changes in covariance. In our second example we attempted to capture only smooth changes in our covariance, and use what was not captured to find anomalies. Our final example shows a 3-D adaptive prediction error output for a synthetic with fine layering, faulting, and a wide range of dips.

## Basket fabric data

Our first example was to check make sure our filter estimation was producing reasonable results. We chose the basket fabric from Claerbout (2014) because it had significant spatial variation that confused stationary filtering. We did not expect to perfectly capture the covariance because the abrupt change in dips in the fabric breaks our fundamental assumption of smooth change implied by our filter interpolation. We first estimated the covariance by solving in a least-squares sense the objective

$$\mathbf{0} \approx \mathbf{D}\mathbf{f}, \tag{3}$$

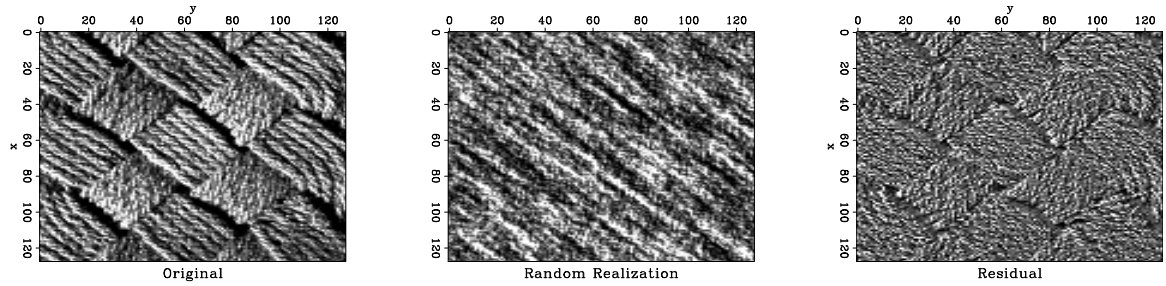
where $\mathbf{0}$ is a vector of zeros, $\mathbf{D}$ is convolution with our data, $\mathbf{f}$ is our inverse covariance PEF estimate. Viewing $\mathbf{f}$ is uninformative. To get a sense of how effective our representation of the covariance was, we attempted to see the inverse of $\mathbf{f}$ by estimating the least squares objective function represented by

$$\mathbf{n} \approx \mathbf{F}\mathbf{m}, \tag{4}$$

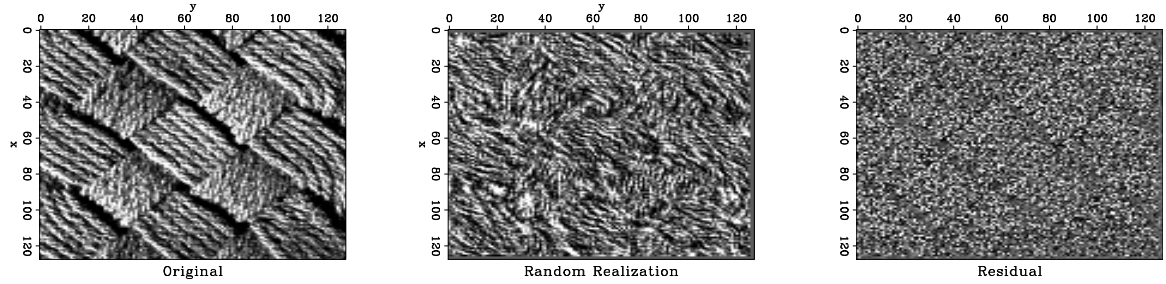
where $\mathbf{n}$ contains random noise. Any random noise vector in equation 4 gives a random realization of data. $\mathbf{F}$ is convolving with our PEF representation of the inverse covariance, and $\mathbf{m}$ is our model that should now have the same covariance (the inverse of the inverse covariance) as our original data. In addition, we inspected the result of filtering our data $\mathbf{d}$ with our PEF representation,

$$\mathbf{r} = \mathbf{F}\mathbf{d}, \tag{5}$$

where **r** should be IID if we perfectly captured the model's spectrum. We estimated the fabric's spectrum using both a stationary (Figure 3(a)) and non-stationary (Figure 3(b)) filter. In each figure the left panel shows the original fabric, the center panel shows **m** from equation 4, and the right panel shows the residual from equation 5. Note how for the stationary case we see the same dips everywhere in the center panel of Figure 3(a), while we see changing dip pattern in the non-stationary case (Figure 3(b)). For the stationary case in the right panel of Figure 3(a), we can clearly delineate the boundaries where the dips changed in the original fabric, while in the non-stationary case these changes are not as obvious. In both cases the residual is approximately the same amplitude. As expected, the non-stationary filter did not do a significantly better job in capturing the fabric's covariance. Looking at the center panel of Figure 3(b) we see it attempted to explain the abrupt changes through smoothly changing the dips. Exactly what we hoped to see.




(a)




(b)

Figure 3: Basket weave: (a) Stationary, (b) Non stationary. A 45° trend dominates. **Realization:** On the stationary we see the 45° trend everywhere. **Residual:** Trends are less pronounced. [**ER**]

## Feature identification

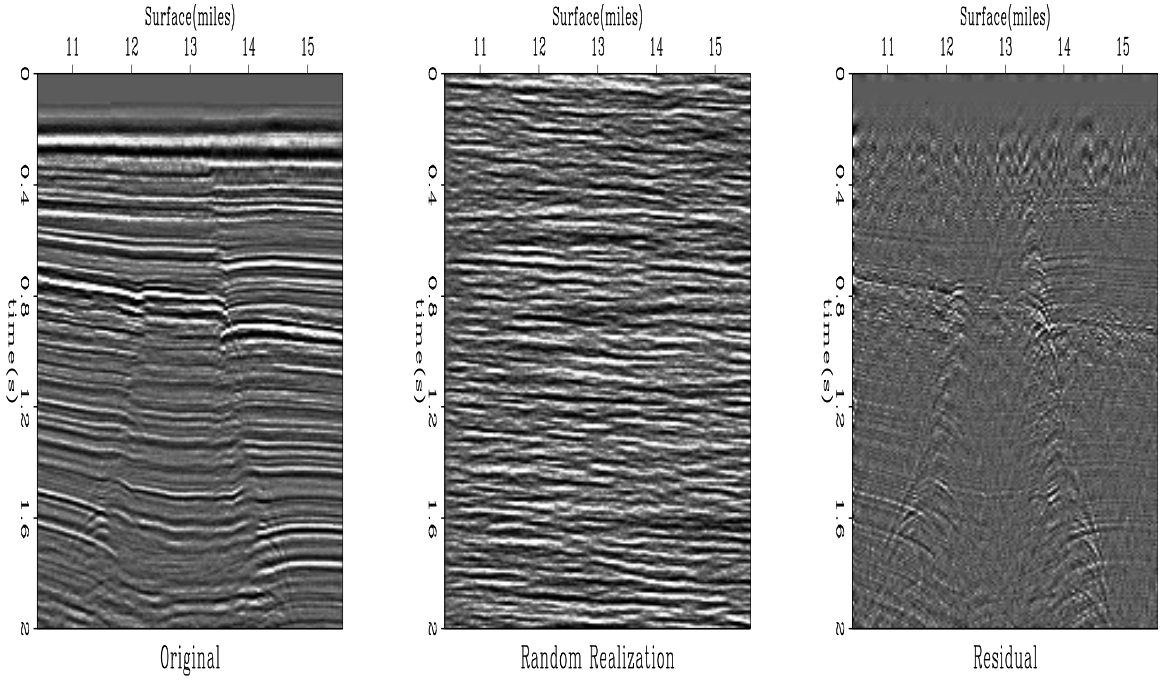
One of the advantages of our approach to estimating non-stationary covariance is that we have fewer adjustable parameters as compared to other approaches. Our approach requires us to define the shape of the PEF and the distance between our control points where the different PEFs sit. Claerbout (2014) gives a fuller description of PEF design, but for this paper two concepts are important. In the stationary case

the length of the filter along the time axis controls the range of dips we can see and how accurately we can capture a wavelet (more points implies finer sampling as a function of frequency). In the stationary case the number of rows defines how many dips we wish to capture. In the non-stationary case these rules generally still hold. With our non-stationary approach we also set the distance between our control points, which generally determines the rate of dip spectrum change we want to capture. The "generally" used above is because these parameters do interact. Adding rows to a PEF will allow it to find additional dips (that might be a function of non-stationarity). As the control points get closer together different PEFs can start to detect different dips that are actually stationary in nature.

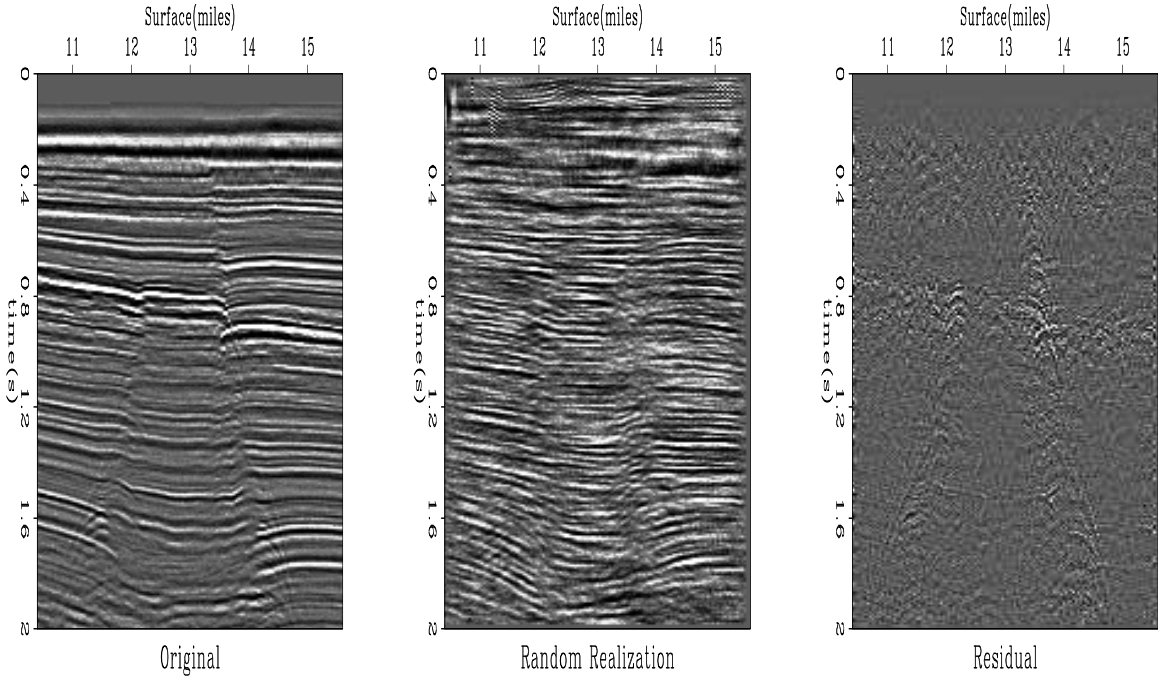
For our second test we wanted to see how well a non-stationary PEF could capture how the spectrum of an image changes in frequency content as a function of time and dip content as function of (mainly) space. We made the choice to *not* try to capture fast changes. To accomplish these goals, we designed our PEF so it was limited in how many dips it could capture, and we put a significant distance between our control points. Figure 4(b) shows the result. Again the left panel show the original data, the center panel the simulated data using equation 4, and the right panel the residual (equation 5). For comparison we did the same test using a stationary filter (Figure 4(a)). If you look at the residual of the stationary result you can still clearly see bedding structures throughout the residual. In the non-stationary residual almost all of the horizontal bedding structure is absent. The simulated data shows why. The non-stationary simulated data have the same changing dip pattern seen in the original image while stationary result only shows the dominant dip direction. As mentioned, we carefully chose our parameters to limit the number of dips we could capture. Note that in the residual of the non-stationary case the diffractions are the dominant feature.

## Qdome 3-D

The Qdome 3-D quarter dome synthetic Claerbout (1993) shown in Figure 5(a) has a wide range of dips and azimuths, including aliased reflectors, and so provides an appropriate challenge for our nonstationary prediction error filtering. The prediction error of Figure 5(b) is a mixed result. On the positive side, it removed a large percentage of the smooth bedding amplitudes and clearly outlined the location of the fault surface. On the negative side, we have left significantly more bedding texture than in our earlier seismic Gulf of Mexico data example and we have, unexpectedly, not handled the steep aliased dips along the left hand side. Deadline pressure did not allow exploring various filter shapes and sizes.

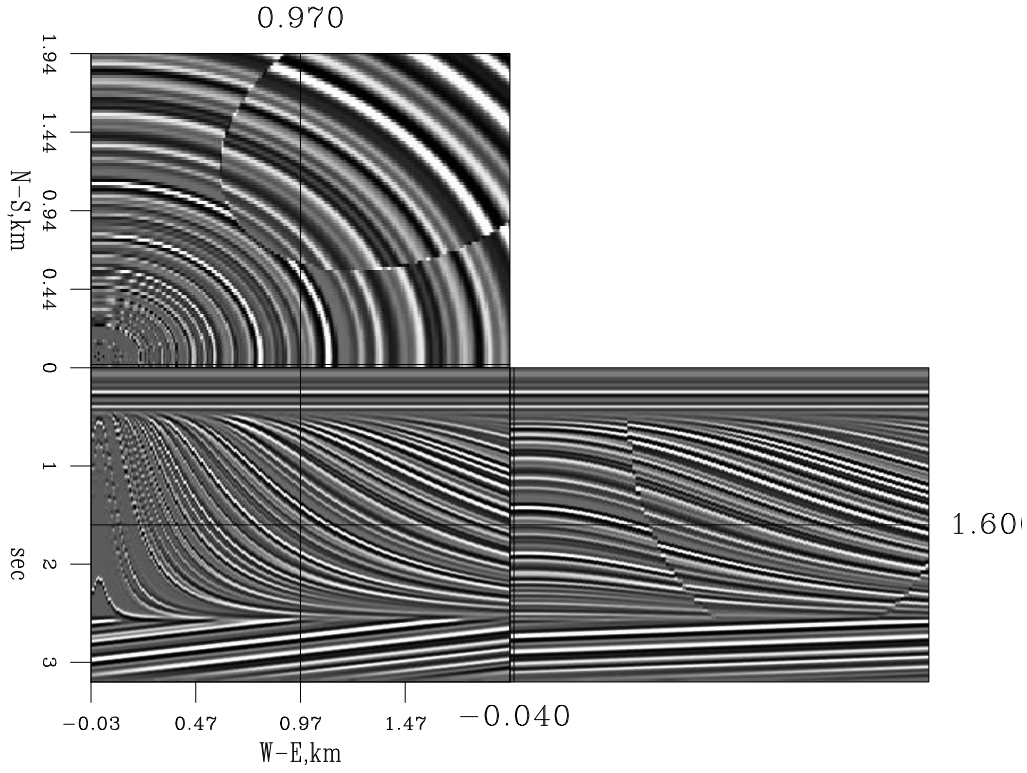



(a)

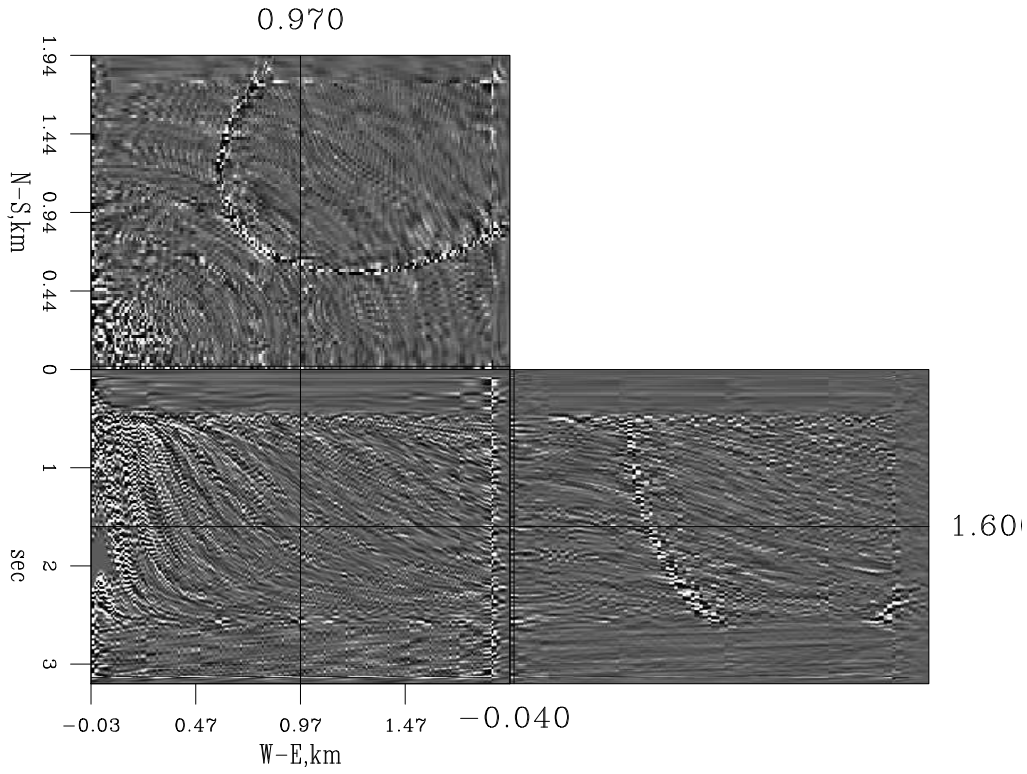



(b)

Figure 4: Gulf of Mexico seismic stacked section: (top) stationary, (bottom) nonstationary. **Random realization:** The nonstationary preserves the fault location because it is embedded in the local PEF. While stationary scatters low frequencies throughout, nonstationary keeps them at early time. **Residual:** The stationary does not drive down the horizontal layers very well because their slopes vary with location. [**ER**]




(a)




(b)

Figure 5: (a) Original Qdome data. (b) Prediction-error residual. The blank borders tell us the size of our $20\times10\times10$ filter stencil. Here the horizontal display exaggeration is 2:1 in $x$ and 4:1 in $y$ relative to the $z$ axis. [**CR**]

## CONCLUSIONS AND DISCUSSION

We presented an addition to the SEP inversion library to capture non-stationary covariance. We tested the library on a series of problems including feature identification and missing data. In the future we will extend this work into even higher dimensional spaces and use it for non-stationary decon and signal and noise separtion.

## REFERENCES

Claerbout, J., 2014, Geophysical Image Estimation by Example: lulu.com.

Claerbout, J. F., 1993, 3-D local monoplane annihilator: SEP-Report, **77**, 19–25.

Crawley, S., 2000, Seismic trace interpolation with nonstationary prediction-error filters: PhD thesis, Stanford University.

Flahive, M., 2008, Balancing R-ary Gray codes II: The Electronic Journal of Combinatorics, **15**, no. R128.

Hale, D., 2006, Recursive Gaussian filters: Center for Wave Phenomena Report, **546**.

Knuth, D. E., 2005, The art of computer programming: Volume 4 Generating all tuples and permutations.

Newton, I., 1687, Philosophiae Naturalis Principia Mathematica: S. Pepys. (Divided differences are outlined in Book III, Lemma V).

Savage, C., 1997, A survey of combinatorial Gray codes: SIAM Review, **39**, no. 4, 605–629.

Schwab, M., 1998, Enhancement of discontinuities in seismic 3-D images using a Java estimation library: **99**.

Stirling, J., 1730, Methodus differentialis: sive tractatus de summatione et interpolatione serierum infinitarum. Auctore Jacobo Stirling, R.S.S.: Gul. Bowyer, London. (Newton's divided difference method described in *Pars Secunda de Interpolatio Serierum, p. 85ff*).