Multichannel data: separating independent causes

Jon Claerbout and Kaiwen Wang

ABSTRACT

The algorithm for blind deconvolution of a nonstationary time series of vector components (*i.e.* multichannel) has three stages: (1) Linear-least-squares multichannel prediction-error filtering, (2) Cholesky factorization of the zero-lag covariance matrix, and (3) Rotation angle scanning for maximum sparsity.

INTRODUCTION

The "blind deconvolution" problem for a vector-valued signal is shown in Figure 1. In practice $x_1(t)$ and $x_2(t)$ may be different wave types that mix and register on our two-component $(y_1(t), y_2(t))$ instruments.

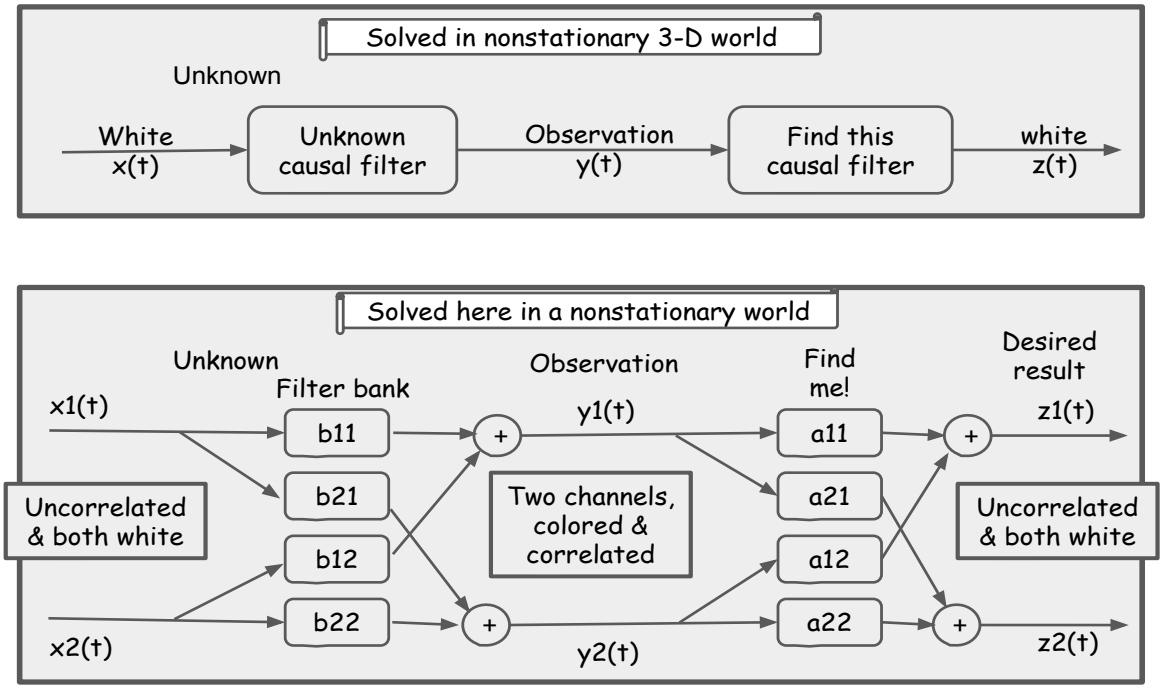

Figure 1: The left side of the diagram hypothesizes nature outside our view. Given the correlated observations $\mathbf{y}$ in the middle we here design the causal process $\mathbf{A}$ to create the outputs $\mathbf{z}$ on the right. We construct uncorrelated white $\mathbf{z}$, then hope it approximates the underlying physical world $\mathbf{x}$, and that $\mathbf{B} \approx \mathbf{A}^{-1}$. [**NR**]

The multichannel structure of Figure 1 arises in diverse physical settings. For example underwater measurements may represent waves upgoing and waves downgoing, but we don't measure them directly; we measure pressure and displacement. Or the earth may contain pressure waves and shear waves while we measure vertical and horizontal motions. Waves may arrive at our multicomponent recoreders from two directions. If your multicomponent recorder records two different things, like pressure and velocity, your channels will have differing spectral characteristics. That's good. It's a central aspect of this model.

Fourier analysis suggests a crude approach to Figure 1. For scalar waves, given the spectrum $Y(\omega)^* Y(\omega)$ the solution to the problem is $A(\omega) = 1/\sqrt{Y(\omega)^* Y(\omega)}$. But this implies a symmetric function of time, not causal. Fourier space requires stationary statistics, forbids ℓ_1-norm. The square root of a matrix of Fourier functions is easily found, but the disadvantages of Fourier space overwhelm the simplicity of the time domain. Causality is easily expressed with Z-transforms, equivalently either a matrix of them, or a polynomial of matrix coefficients.

Theory behind Figure 1 appeared in engineering literature half a century ago. I gave it little attention because I had no data of vector-valued signals. The old theory for such signals also depended on the geophysically unrealistic assumption of stationarity. My google search for terms like "adaptive multichannel filter theory" did not turn up methods recognizably suitable for my community of geophysical data analysts (although I believe it should be out there somewhere in the engineering literature). Then I stumbled onto appropriate methodology for non-stationary signals. It is much easier to put into practice than the stationary theory. Hooray! Our research group began getting multicomponent (vector-valued) data. Here I put all the pieces together and am looking for people who want to try it.

The first stage is multichannel prediction-error theory. Here the approach to matrices of filters learns from the old theory of stationary methods (but nonstationarity makes it *much* easier). PEFs (prediction-error filters) remove all lagged correlations from the data. But at zero lag there remains crosscorrelation between the channels. That's easily dealt with by the Cholesky method.

Intriguing is what comes last, something wholly unfamiliar. Even after solving the problem posed in Figure 1, the solution is unique only within an arbitrary unitary matrix. But this two channel problem, although nonlinear, is easily amenable to a one-parameter exhaustive search. That search can be done to maximize sparsity of the final signals. We humans love the simplest representation of our data. This should be it. Hooray!

In related good news it looks like this final unitary process could overcome a perennial problem in geophysical data processing. Since the processing stream is inexpensive it is easy to program a wide-ranging exploration of the process parameter space. The sparsity measure could then tell how to choose the best processing parameters. Hooray again! A complete, but untested code, for the only subtle part of the whole process, the vector-signal PEF, is found at the end of this document.

Range of applicability

This two-component signal model is not suitable for two scalar signals recorded at separate locations. If you are processing a string of multicomponent recorders (down a well, for example) each multicomponent recorder yields statistics which may be shared and averaged with neighboring recorders, but the signals themselves would not mix, so, given these statistics, the process described here is simply a time-variable linear operator. This mathematical model is based on causality and simultaneity of the two channels responding to the outside world.

The model here naturally extends to three and more physical dimensions. Whether it is suitable for many-channel market signals I cannot say.

If the underlying model $\mathbf{B}$ were to introduce delay, its hypothetical inverse $\mathbf{A}$ would need to contain inverse delay (non-causality!). Then the methods of this paper must fail. In marginal cases (tiny delay) the notion of sparsity has helped for scalar signals. Its a promising area beyond our present scope.

SCALAR NONSTATIONARY SIGNALS

To my surprise and delight, I find nonstationary signal analysis simpler and more amenable to practice than textbook stationary theory.

Nonstationary prediction without mathematics(!)

Start with data (a signal of thousands of values). Take any filter $\mathbf{a}$ of maybe ten lags. The filter will change slowly as we slide it along the data. Set the filter down on the data. The ten data values under the filter are designated $\mathbf{d}$. Take ϵ to be a tiny scalar (for example $\epsilon = 1/(4000\,\|\mathbf{d}\|)$).

At any one time instant the filter $\mathbf{a}$ has output $(\mathbf{a}\cdot\mathbf{d})$. We set the goal that $(\mathbf{a}\cdot\mathbf{d})$ should be something we have chosen. We might have chosen another signal (shaping), or the next incoming data value (prediction). The filter output $(\mathbf{a}\cdot\mathbf{d})$ over-shoots or undershoots this goal. We will add or subtract a teeny ϵ amount of data $\mathbf{d}$ to the filter $\mathbf{a}$, then see which of the $\pm$ does the better job. Trying the modified filter $\mathbf{a}\pm\epsilon\mathbf{d}$ two possible predictions emerge.

$$(\mathbf{a}\pm\epsilon\,\mathbf{d})\cdot\mathbf{d} \quad = \quad (\mathbf{a}\cdot\mathbf{d})\pm\epsilon\,(\mathbf{d}\cdot\mathbf{d}). \tag{1}$$

Comparing these predictions to the hoped for value reveals which sign for ϵ better improves our filter. Update the filter $\mathbf{a}$. Move to time $t+\Delta t$. Update $\mathbf{d}$. Repeat indefinitely. The filter adapts to make the best fit your goal (often prediction, more often prediction error).

The above idea can be based on conventional math. The gradient of ℓ_2 normed prediction error $\mathbf{e}$ turns out to be $\mathbf{d}\times\text{PredictionError}(\mathbf{d})$. Let $\text{signum}(e)=e/|e|$. Then

the above algorithm amounts to stepping along with $\mathbf{d} \times \text{signum}(\text{PredictionError}(\mathbf{d}))$ which smells like nonstationary ℓ_1 norm decon. Wow!

Many variations of this theme are easy, such as constrained filters, gapped filters, predicting further ahead, or predicting other signals. We mostly use these ideas for making prediction error (a white signal). We make that by constraining the first filter coefficient to be +1 so all the rest are effectively predicting negatively to try extinguish the data value under the +1. In the limit of very many iterations and $\epsilon \rightarrow 0$ the result tends to that of stationary theory.

Any color found in the prediction-error output should have been usable to enhance the prediction. Presuming it did, the result is optimal prediction-error output. This tends to whiteness, being limited only by the number of filter coefficients and the non-zero size of ϵ.

Of academic interest the relation between the ℓ_1 norm and the ℓ_2 norm methods is easy suggesting that a wide range of other norms and penalty functions are easily attained in the same way. The heart of the matter seems to be choosing any function of the components of $\mathbf{d}$ that is polarity preserving (so when dotted into $\mathbf{d}$ assures a positive scalar). So it seems the $\ell_{1/2}$ norm is as easy as augmenting the filter $\mathbf{a}$ with a vector of components $\epsilon\, d_i/|d_i^{3/2}|$. Naturally my favorite is the softclip function, the derivative of a hyperbolic penalty function.

Pseudo code for scalar signals

In all least-squares data fitting, the residual goes into the adjoint to produce the gradient direction. Stripped of details, ℓ_2-norm scalar-signal code is

```
a(1) = 1.0           #               Syntax:   "a+=b" means "a=a+b"
do for all time t    # e = nonstationary prediction error
        do tau= 1, na
                        e(t)    +=  a(tau) * y(t-tau+1)      # forward
        do tau= 2, na
                        da(tau) +=  e(t)   * y(t-tau+1)      # adjoint
        do tau= 2, na
                        a = a - epsilon * da
```

Arrays in Fortran/Matlab range from a(1) to a(n) while in C/C++/Java they range from a(0) to a(n-1). Matrix operations are more naturally expressed in Fortran/Matlab while polynomial and convolution operations are more naturally expressed in C/C++/Java. While it may be natural to express each concept in its favored language, my pseudo codes were confusing until I stuck to just one language convention. Since I chose Fortran/Matlab the math idea $\int_0^\infty a(\tau)y(t-\tau)d\tau$ is rendered `do tau=1,na{a(tau)*y(t-tau+1)}`.

The `#forward` code line computes the prediction residual `e(t)` at some time `t`. At that time, the first `tau` loop is performing the dot product $(\mathbf{a} \cdot \mathbf{d})$ mentioned

earlier with **d** being a backwards running chunk of data `y(t-tau)`. The `#adjoint` loop corresponds to a matrix transposed because compared with the `forward` loop, input and output have swapped their roles. Their `tau` loops have differing ranges. This because the `a(1)=1.0` produces the prediction *residual,* so `a(1)` is not changed.

Linguistically, it would be more correct to call $(-1, \cdot, \cdot, ...)$ the prediction-error filter, but nobody wants to use that filter because its output polarity is opposite that of the original data. So the PEF is defined as $(+1, a_1, a_2, ...)$.

Gradient derivations

Formal theory underlies the idea of adding an ϵ bit of **d** to the filter. The algebra to show that **d** $\times$ PredictionError(**d**) is the gradient that arises from a new data point is in Appendix I.

Sergey Fomel and I developed a complicated, subtle nonstationary PEF theory. It led to an update direction with a conceptual distance parameter. Then I discovered its implementation amounts to a simple-minded step in the direction of the gradient. Operationally, the two methods had turned out to be one and the same! The "simple minded conceptual parameter" amounts to our epsilon ϵ.

Easy question: Given that seismic data is typically gained with t^2, how should gain and decon work together?

How big is epsilon?

Epsilon ϵ is the fractional change to the filter at each stage of iteration. In a process called leaky integration, any value at time t is altered by a fractional amount ϵ during transition to $t + \Delta t$. Every smoothed value is diminished by $(1 - \epsilon)$, and then updated by ϵ times its current estimated value. After λ steps any value is reduced by the factor $(1 - \epsilon)^\lambda$. Setting that to $1/e = 1/2.718$ says $1/e = (1 - \epsilon)^\lambda$. Taking the natural logarithm, $-1 = \lambda \ln(1 - \epsilon) \approx -\lambda\epsilon$, so to good approximation

$$\epsilon \;=\; 1/\lambda \tag{2}$$

By the well known property of exponentials, half the area in the decaying signal appears before the distance λ, the other half after.

In casual discussion I think of this memory function as a rectangle function of length λ. Least squares analysis begins with the idea there should have more regression equations than unknowns. So λ should roughly exceed the number of filter coefficients `na`. To avoid overfitting, I'd begin with $\lambda = 10\times$`na`.

I have more thoughts on choosing λ, but this is too early to present them. The nonstationary environment is such a strong component of many valuable applications that reports of wise and clever choices for ϵ, if they cannot be found now, they are sure to arise soon.

VECTOR NONSTATIONARY SIGNALS

In scalar signal analysis, it is known that the prediction-error signal $e(t)$ is white Claerbout (2014) (page 182). Its autocorrelation is a delta function. Something similar (but intriguingly different) happens with vector-valued signals. Vector and scalar cases are are based on causality. Innovations arrive simultaneously on both channels. The two channels may show different spectra, but the method fails when one component has been delayed with respect to the other, so don't try it with two scalar channels recorded at different locations.

Nonstationary vector-valued signals require a three-stage process. The first stage mimics the prediction-error process of scalar signals. That eliminates time-lagged correlations. The next stage, the Cholesky stage, eliminates zero-lagged crosscorrelation between the two channels; and it scales the channels to unit variance. At the last stage notice the vector process defined by Figure 1 has multiple solutions. From any vector solution, others follow by any unitary matrix $\mathbf{U}$ tranformation. (With scalar-signals the arbitrariness is in a scale factor $e^{i\phi}$.) We get to choose the $\mathbf{U}$ having minimum entropy $\mathbf{z}$ output. Unexpected. Intriguing!

Nonstationary variance/covariance

Equation (3) defines a running variance $\sigma_y^2(t)$ of the signal $y(t)$. Such a recursive process is called leaky integration. Likewise (4) defines a running crosscorrelation between two channels.

$$\sigma_y^2(t) = (1-\epsilon)\,\sigma_y^2(t-\Delta t) + \epsilon\, y(t)^2 \tag{3}$$

$$\sigma_{y_{12}}^2(t) = (1-\epsilon)\,\sigma_{y_{12}}^2(t-\Delta t) + \epsilon\, y_1(t)\, y_2(t) \tag{4}$$

$$\sigma_{\mathbf{y}}^2(t,\tau) = (1-\epsilon)\,\sigma_{\mathbf{y}}^2(t-\Delta t) + \epsilon\, \mathbf{y}(t)\, \mathbf{y}(t+\tau)' \tag{5}$$

Likewise (5) defines a 2×2 matrix of running lagged covariance where $\mathbf{y}(t)$ is a two-component column vector while $\mathbf{y}(t+\tau)'$ is a likewise a row. Stationary time-series theory actually displays the 3-D lagged covariance $\sigma_{\mathbf{y}}^2(\tau)$ (FGDP page 140).

Our 2-component PEF marching along the time axis updating prediction-error filters should chew up the lagged correlations. That provably happens with scalar signals Claerbout (2014) (page 182). Thus the PEF output $\mathbf{e}(t)$ has a 3-D covariance that vanishes at nonzero lags. We are left with the zero lag, a nice 2×2 matrix of prediction-error variances $\mathbf{W}$.

$$\mathbf{W}(\tau=0) = \begin{bmatrix} \sigma_{e_{11}}^2 & \sigma_{e_{12}}^2 \\ \sigma_{e_{21}}^2 & \sigma_{e_{22}}^2 \end{bmatrix} \approx \begin{bmatrix} (\mathbf{e}_1 \cdot \mathbf{e}_1) & (\mathbf{e}_1 \cdot \mathbf{e}_2) \\ (\mathbf{e}_2 \cdot \mathbf{e}_1) & (\mathbf{e}_2 \cdot \mathbf{e}_2) \end{bmatrix} \tag{6}$$

The dot products are an oversimplification intended to clarify the meaning of leaky integration to new users. The dot products are also a handy way to initialize the update expressions.

Scalar signal scaling

A length measurement $\lambda = 1/\epsilon$ (in pixels) measures the averaging region. Thus both λ and ϵ are without physical units, though one might say λ has "units" of pixels. Prediction filters are dimensionless because from voltage, they predict voltage. Hence the units of $e\mathbf{d}$ match those of σ_d^2. The properly scaled ℓ_2 update expression is

$$\Delta\mathbf{a} \quad = \quad - \left(\frac{\epsilon \; e}{\sigma_d^2} \right) \; \mathbf{d} \tag{7}$$

Replacing the prediction error e by its signum function yields an ℓ_1-norm prediction after restoring nondimensionality by changing σ_d^2 to σ_d.

$$\Delta\mathbf{a} \quad = \quad - \left(\frac{\epsilon \; \text{signum}(e)}{\sigma_d} \right) \; \mathbf{d} \tag{8}$$

Let σ_e be a running standard deviation of prediction error. Now I'm feeling an ℓ_2-norm filter update slightly more consistent than equation (7) is

$$\Delta\mathbf{a} \quad = \quad - \left(\frac{\epsilon \; e}{\sigma_e \sigma_d} \right) \; \mathbf{d} \tag{9}$$

Understanding physical units with scalar signals leads next to vector signal scaling.

Vector signal scaling

The leading coefficient of a vector signal PEF (prediction-error filter) is an identity matrix. The two 1's in the $\mathbf{I}$ pass through the observed data $\mathbf{y}(t)$. Coefficients under all the other lags adapt to negatively predict it so as to get minimal output.

When components of data or model are out of scale with one another, bad things happen: The adjoint operator will not be a good approximation to the inverse. Physical units may be contradictory. Steepest descent creeps along slowly. These dangers would arise with vector-valued signals if the observations y_1 and y_2 had different physical units such as pressure and velocity recorded from up- and down-going waves. Or such as uncalibrated vertical and horizontal seismograms.

One could devise the filter updates by an effort of deep thought while inspecting Figure 1, but it's easier and more reliable to blindly base your updates on the negative adjoint of forward modeling. But, we do need to think about channels being out of scale with one another. Thus we scale each component of data $\mathbf{y}$ and residual $\mathbf{e}$ by dividing out their variances as we did in equation (9). Recall that any component of a gradient may be scaled by any positive number. Such scaling is merely a change in coordinates.

This is a good time to read the code at the end of this article.

Averaging in time and space

The code contains leaky integrations to assure the filter $\mathbf{A}$ varies smoothly in time. Actually, the leaky integrations may smooth over both time and space. In other words, when updating an old filter $\mathbf{A}(t-\Delta t, x)$, we could update the old filter located at $\mathbf{A}(t, x-\Delta x)$. That would be learning over x while filtering over t. More generally, an update could leap from a weighted average over time and space. For example, we could update $\mathbf{A} \leftarrow \overline{\mathbf{A}} + \Delta\mathbf{A}$ with

$$\overline{\mathbf{A}} \quad = \quad \mathbf{A}(t-\Delta t, x)\ \frac{\lambda_t^2}{\lambda_t^2+\lambda_x^2} \ + \ \mathbf{A}(t, x-\Delta x)\ \frac{\lambda_x^2}{\lambda_t^2+\lambda_x^2} \tag{10}$$

Notice that the weights sum to unity. The averaging region is an area roughly $\lambda_x\lambda_t$ pixels squared in size. The coding requires not only saving $\mathbf{A}$ at the previous time, it requires a saved $\mathbf{A}$ for every time at $x-\Delta x$.

Stationary decon should remove a shot waveform. Nonstationary decon starts from there but has the added possibility of removing the waveform of the outgoing wave. That evolves with traveltime (Q and forward scattered multiples). It also evolves with space, especially receiver offset. We could build such nonstationary filters on either field data or synthetic data, then apply them to field data. The relations among pressure, velocity, upcoming, and downgoing waves vary systematically with offset. You could work out theoretical expressions for these relations, but instead you could see how this data fitting code would handle it.

How can the nonstationary PEF operator be linear?

Let $\mathbf{E}$ be the prediction-error operator and $\mathbf{e}$ its output. By definition

$$\mathbf{e} \quad = \quad \mathbf{E}\,\mathbf{y} \tag{11}$$

The operator $\mathbf{E}$ may seem to be a nonlinear function of the data $\mathbf{y}$. But it is nearly linear, even strictly linear in a certain sense. Notice that $\mathbf{E}$ could have been built entirely from spatially nearby data, not at all from $\mathbf{y}$. Then $\mathbf{E}$ would be nonstationary, yet a perfectly linear operator on $\mathbf{y}$.

I am no longer focused on conjugate-gradient solutions to stationary linear problems, but if I were, I could at any stage make two copies of all data and models. The solution copy would evolve with iteration while the other copy would be fixed and would be used solely as the basis for PEFs. Thus the PEFs would be changing with time while not changing with iteration. This makes the optimization problem a linear one, fully amenable to linear methods. In the spirit of conjugate gradients (as it is commonly practiced), on occasion we might restart with an updated copy. People with inaccurate adjoints often need to restart. (Ha ha.)

CHOLESKY DECORRELATING AND SCALING

The two independent channels of unit-variance random numbers in $\mathbf{x}$ *entering* filter $\mathbf{B}$ in Figure 1 have the identity matrix $\mathbf{I}$ as a covariance. Here we arrange to have the same identity covariance for the values $\mathbf{z}$ *exiting* from $\mathbf{A}$ on the right.

Consider the expectation (leaky sum over time) $\mathrm{E}[\,\mathbf{e}\,\mathbf{e}']$. Theoretically it's a 3-D function of lag and the two channels. We're going to assume our PEFs are perfect so that it is no longer a function of lag. Thus we presume that $\mathrm{E}[\,\mathbf{e}\,\mathbf{e}']$ is like the $\mathbf{W}$ we computed with equation (6) at zero lag τ.

$$\mathrm{E}[\,\mathbf{e}\,\mathbf{e}'] \quad = \quad \begin{bmatrix} \sigma_{e_{11}}^2 & \sigma_{e_{12}}^2 \\ \sigma_{e_{21}}^2 & \sigma_{e_{22}}^2 \end{bmatrix} \quad = \quad \mathbf{W} \tag{12}$$

Use the Cholesky method to factor $\mathbf{W}$ into a triangular matrix $\mathbf{V}$ times its transpose, so $\mathbf{W} = \mathbf{V}\mathbf{V}'$. (The Cholesky method is nearly trivial: Write a triangular matrix of unknown elements. Multiply it by its transpose. Notice a sequential method that unravels the unknown elements.)

$$\mathbf{W} = \mathbf{V}\,\mathbf{V}' \tag{13}$$
$$\mathbf{V}^{-1}\mathbf{W}(\mathbf{V}')^{-1} = \mathbf{I} \tag{14}$$
$$\mathbf{C}\mathbf{W}\mathbf{C}' = \mathbf{I} \tag{15}$$

where we have defined $\mathbf{C} = \mathbf{V}^{-1}$. Using this new matrix operator $\mathbf{C}$ we get a new vector signal $\mathbf{q}$.

$$\mathbf{q} \quad = \quad \mathbf{C}\,\mathbf{e} \tag{16}$$

The expectation of this new variable $\mathbf{q}$ is

$$\mathrm{E}[\mathbf{q}\mathbf{q}'] = \mathrm{E}[\mathbf{C}\,\mathbf{e}\mathbf{e}'\,\mathbf{C}'] \tag{17}$$
$$= \mathbf{C}\,\mathrm{E}[\mathbf{e}\mathbf{e}']\,\mathbf{C}' \tag{18}$$
$$\mathrm{E}[\mathbf{q}\mathbf{q}'] = \mathbf{C}\,\mathbf{W}\,\mathbf{C}' \quad = \quad \mathbf{I} \tag{19}$$

This shows Cholesky does for us two things: (1) it descales, and (2) it decorrelates $\mathbf{e}$ at zero lag.

ROTATING FOR SPARSITY

The most intriguing part of the entire process arrives at this the last stage. As the universe marches on, things get mixed and entropy increases. We seek the opposite.

Rotations and reflections are called unitary operators. For now we are ignoring reflections (polarity changes). (Consider that to be an application labeling issue.) Scanning a single parameter θ through all angles allows us to choose the one with the most sparsity (least clutter). A general form for a 2×2 rotation operator is

$$\mathbf{U} \quad = \quad \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{20}$$

We will meet our goal of finding $\mathbf{A}$ and $\mathbf{z}$ of Figure 1 with:

$$\mathbf{z} = \mathbf{U}\mathbf{q} = \mathbf{U}\mathbf{C}\mathbf{e} = \mathbf{U}\,\mathbf{C}\,\mathbf{E}\,\mathbf{y} = \mathbf{A}\,\mathbf{y} \tag{21}$$

A unitary operator $\mathbf{U}$ does not change the length of any vector. It satisfies $\mathbf{U}'\mathbf{U} = \mathbf{I}$, so for any $\mathbf{v}$ we see $(\mathbf{U}\mathbf{v})'\mathbf{U}\mathbf{v} = \mathbf{v}'\mathbf{U}'\mathbf{U}\mathbf{v} = \mathbf{v}'\mathbf{v}$. Let us check that the covariance of $\mathbf{z} = \mathbf{U}\mathbf{q}$ is constant independent of θ. Equation (19) leads to

$$\mathbf{z}\mathbf{z}' \quad = \quad \mathbf{U}\ \mathrm{E}[\mathbf{q}\mathbf{q}']\ \mathbf{U}' \ = \ \mathbf{U}\mathbf{I}\mathbf{U} \ = \ \mathbf{I} \tag{22}$$

This is saying the energy stays constant as we sweep through θ.

Finding the angle of maximum sparsity (minimum entropy)

Given any angle θ for equation (20) we have $\mathbf{z} = \mathbf{U}\mathbf{q}$. We can scan θ over one degree increments. Defining the entropy at any particular time as $(|z_1| + |z_2|)/\sqrt{z_1^2 + z_2^2}$ we easily choose the angle of minimum entropy for that time.

The more difficult question is dealing with noise. We want estimates based on time averages. Ultimately, we will have arrays of vector valued signals. We will also want local averages in the space of the arrays. At the deadline for this progress report, we are not certain we have have properly dealt with the issue of estimating a best angle by forming averages over time and space.

Jon's theory, apparantly defective

The code below is a guess at the solution to the problem of averaging the Cholesky output $\mathbf{q}(t)$ over time to finally find a best angle for rotation. I define z_1 and z_2 by leaky integrating over time t the four quantities $q_1(t)\cos\theta$, $q_2(t)\sin\theta$, $q_1(t)\sin\theta$, $q_2(t)\cos\theta$, which contain all the parts of the product of vector $\mathbf{q}$ multiplying the matrix of (20). Then I define entropy at time t by $(|z_1| + |z_2|)/\sqrt{z_1^2 + z_2^2}$. Finally, I scan all angles for the minimum entropy, and choose that angle θ.

```
initialize {q1cos,q2sin,q1sin,q2cos}(1:360)=0, entropy(1:360)=0
do over all time t {
   # You insert steps "q = C E y" here.
   do ith= 1, 360 {
      th = 2 * 3.1416 * (ith-1)/360.
      q1cos(ith) = (1-epsilon)*q1cos(ith) + epsilon*( q1(it)*cos(ith))
      q2sin(ith) = (1-epsilon)*q2sin(ith) + epsilon*( q2(it)*sin(ith))
      q1sin(ith) = (1-epsilon)*q1sin(ith) + epsilon*( q1(it)*sin(ith))
      q2cos(ith) = (1-epsilon)*q2cos(ith) + epsilon*( q2(it)*cos(ith))
      z1 =   q1cos(ith) + q2sin(ith)
      z2 =  -q1sin(ith) + q2cos(ith)
      entropy(ith) =  (abs(z1) + abs(z2)) / sqrt( z1*z1 + z2*z2 )
      }
```

```
   ithbest = 1        # Find the best theta
   do ith= 1, 360
       if( entropy(ith) < entropy(ithbest)) ithbest = ith
   # You put theta(ithbest) into the U matrix and make "z(t) = U C E y(t)"
   }
```

Kaiwen's theory: works on easy synthetics

The code below is to find a best angle for each time step to rotation. It follows Stew Levin's suggestion to apply phase unwrapping afterwards to avoid switch or flip of trace. We define u_1 and u_2 by leaky integrating over time t the l_1 and l_2 norm of (z_1, z_2). Then We define entropy at time t by u_1/u_2. Finally, We scan all angles for the minimum entropy, and choose that angle for the wrapped θ. We see that the period of θ is $\pi/2$, so we choose the jump tolerance to be $\pi/4$ and correct phase change.

Matlab code:

```
u1=zeros(360,1);
u2=zeros(360,1);
entropy=zeros(360,1);
for i=na+1:nt
for ith=1:360
        theta=2*pi*(ith-1)/360;
        u1(ith)=(1-epsilon)*u1(ith)+epsilon*norm( ...
      [q(1,i)*cos(theta)+q(2,i)*sin(theta) -q(1,i)*sin(theta)+q(2,i)*cos(theta)],1);
        u2(ith)=(1-epsilon)*u2(ith)+epsilon*norm( ...
      [q(1,i)*cos(theta)+q(2,i)*sin(theta) -q(1,i)*sin(theta)+q(2,i)*cos(theta)]);
       entropy(ith)=u1(ith)/u2(ith);
       end
    [~,I]=min(entropy);
    theta=2*pi*(I-1)/360;
    theta_wrapped(i)=theta;
end
theta_unwrapped=theta_wrapped;
for j=2:length(theta_unwrapped)
    difference = theta_unwrapped(j)-theta_unwrapped(j-1);
    while abs(difference) > pi/4
        if difference > pi/4
            theta_unwrapped(j:end) = theta_unwrapped(j:end) - pi/2;
        elseif difference < -pi/4
            theta_unwrapped(j:end) = theta_unwrapped(j:end) + pi/2;
        end
        difference = theta_unwrapped(j)-theta_unwrapped(j-1);
    end
end
for i=na+1:nt
    U=[cos(theta_unwrapped(i)) sin(theta_unwrapped(i)); ...
      -sin(theta_unwrapped(i)) cos(theta_unwrapped(i))];
    z(:,i)=U*q(:,i);
end
```

Why the scan works

Why does this $\mathbf{U}$ process of scanning θ lead to sparsity? Suppose the vector signal element $\mathbf{q}_N$ at time at $t = N$ has all its energy in its first component. Say the vector signal is $[-1, 0]'$ with energy and magnitude both now equal unity. The rotated signal is now

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad = \quad \begin{bmatrix} -\cos\theta \\ \sin\theta \end{bmatrix} \tag{23}$$

Let the rotation angle be 45° so sine and cosine are both $1/\sqrt{2}$. The sum of the magnitudes becomes $2/\sqrt{2} = \sqrt{2} > 1$. As expected the rotation took away the original sparsity.

3-component vector data

For 3-component vectors the scan would run over two angles so the `u(itheta)` would be expanded to `u(itheta,iphi)`.

NOT QUITE MINIMUM DELAY

We have solved the spectral factorization problem for nonstationary vector-valued signals. It might seem to be a unique solution. But it is not. If the physics $\mathbf{B}$ includes delays, we won't find their inverse delays in our computed $\mathbf{A}$ because our $\mathbf{A}$ is causal. Presuming that $\mathbf{z} = \mathbf{x}$ amounts to making the so-called "minimum-phase" assumption for vector-valued signals. I don't know the general form for matrix causal all-pass (pure delay) filters but I do know it for scalar filters, and it suggests a very big world of delay possibilities.

But tiny delays are not uncommon in practice and may be overcome. Particularly instructive is my experience with scalar signals:

Deconvolution has been an industry standard for half a century. Conventionally, it is an ℓ_2-norm process that makes a minimum-phase assumption. This means the inverse filter should be causal. It is obvious to everyone that the shot waveform is causal, but that should not be confused with its inverse. A dangerous ingredient is the marine ghost. Its heart is a second-difference operator δ_{tt} so its inverse involves a growing ramp! So, like many others, I set out to look for a causal shot waveform with a noncausal inverse. This isn't easy, but with Antoine Guitton (and earlier work with Yang Zhang and Yi Shen) we invoked the concept of sparsity. To do so takes us beyond the ℓ_2 norm. We got amazing results:

On all the data tested, four field data sets, and a fifth modeled at a sponsoring company, we were able to show images having events with signal polarity that was self evident. The first multiple would have a polarity obviously opposite the primary.

A white reflection denoted a hard reflector, a black one denoting a soft one, so bottom of salt was easily recognizable by the polarity change. I found it thrilling be able to view many other events on seismic data in such a geologic manner.

MISCELLANY

Avoiding near Nyquist confusion

PEFs seem to focus equally on all frequencies up to the Nyquist. Sometimes analysts choose to focus more on the lower range. This can be done by "gapping", namely, constraining to zero a few filter coefficients near the leading 1.

Leaky integration alternatives

The choice of an area in which to gather statistics is fairly subjective. From equation (10) and the leaky integration expression we may wonder the contours of the 2-D integration response. Contours of the product $e^{-t/\lambda_t - x/\lambda_x}$ in (t, x) are a simple triangle, so other than the sharp corner, it feels reasonable though very basic.

Sometimes I feel I'd be more comfortable if I could say statistics had been uniformly weighted before decay sets in. I could easily make exact boxes with $(1 - Z^N)/(1 - Z)$ but I don't like the sharp truncation after N lags. I also don't like the sharp corner at the beginning of the exponential. Perhaps something like a box $(1 - Z^N)/(1 - Z)$ terminated by a damped exponential $+Z^N/(1 - (1 - \epsilon)Z)$. Those two outputs could simply be added. Alternately, those two polynomial ratios could be added, then rearranged to the single ratio $(1-\rho Z-(1-\rho)Z^{(N+1)})$ / $(1-(1+\rho)Z+\rho Z^2)$. Multiply the numerator times the Z transform of x_t, namely $X(Z)$, and the denominator by $Y(Z)$, and identify time lags with powers of Z obtaining the recursion:

$$y_t = (1 + \rho)y_{t-1} - \rho y_{t-2} + x_t - \rho x_{t-1} - (1 - \rho)x_{(t-(N+1))} \tag{24}$$

For $N = 2$, $\rho = .5$, and $x_3 = .99$, I obtain y_t = .00 .00 .99 .99 .99 .49 .25 .12 .06 .03.

TEST CASES

To compare inputs with outputs, display as 50 channels per sheet of wiggle trace in 10 groups of 5, the 5th being a dead trace to clarify display.

Easiest case, mixing, but no filtering

```
polarity = 1.
do i=1,1000,40  {   # jump in steps of size 40
```

```
x1(i) = 2.
x2(i+20) = polarity
polarity = -polarity
}
```

$$\mathbf{B} \quad = \quad \begin{bmatrix} 1 & -.3 \\ .2 & 1 \end{bmatrix} \tag{25}$$

The unscrambling would all be done by Cholesky and Unitary.

A case with an obvious answer

I'm not sure the method of this paper should unscramble it. I'm not really sure what these methods should be capable of, but this one should be really impressive if it works.

$$\mathbf{B} \quad = \quad \begin{bmatrix} 1./(1-.6Z) & -.3/(1-.9Z) \\ .2/(1-.6Z) & 1./(1-.9Z) \end{bmatrix} \tag{26}$$

For more fun, we might prefer wavelets that oscillate.

Horizontal phones, two far away signals, near each other, one stronger, the other $5\times$ *weaker coming in at slightly different angles*

$$\mathbf{x} \quad = \quad \begin{bmatrix} S \\ W \end{bmatrix} \quad = \quad \begin{bmatrix} \text{random numbers} \\ \text{random numbers} \end{bmatrix} \qquad \text{for i} = 1, 10000 \tag{27}$$

$$\mathbf{B} \quad = \quad \begin{bmatrix} 5/(1-.6Z) & 1/(1-.9Z) \\ 4/(1-.6Z) & 1/(1-.9Z) \end{bmatrix} \tag{28}$$

SUGGESTIONS AND RANDOM THOUGHTS

Gapped filters for oversampled data

Seismic data is typically oversampled on time. This implies that relatively, the upper half of the bandwidth may be mostly noise. Could we improve the wave-type separation by altering it in some way? We might design PEFs for prediction distances greater than one Δt sample.

A non-causal PEF approximation

A PEF is causal, has a white output, and is able to fit any spectrum. Extending that PEF to allow some negative lags quickly loses the white-out aspect, and we would be

using more coefficients than needed to represent any spectrum. At the same time, we'd like to extend the PEF into negative lags to enable it to attack the δ_{tt} inherent to most seismograms. So, we'd like a few negative lags and we are often happy to give up a few positive lags. With non-stationary methods, testing has become much easier. We should try out the filter $(a_{-2}, a_{-1}, 1, 0, 0, a_3, a_4, a_5, \cdots)$.

Four component data

Some data is sampled with four components, three velocity directions and pressure. This process would output four channels. This sounds redundant. What might we learn from the weakest channel?

Step sizes

How large should be the down-gradient hops? One first guess is to compare successive hops. The polarity of the dot product of the two successive gradients gives the indicator. If they are in the same direction, we might increase the step size. If in opposite directions, we would then decrease it. Is this a sensible strategy? If so, it could be widely used broadly in data fitting, in applications having little to do with statistics. A diverse collection of tests could be intriguing.

In fitting seismic waveforms to field data, we know the data is 95% repeatable, while we are astonished if any theory can drive the fitting residual down by as much as 50%. Does it make sense to struggle with second derivatives whose purpose is to define the location of perfect fit? Shouldn't we focus on more efficient ways of driving down the gradient? For example, gradients are functions of locations in space and frequency, so instead of looking for a global distance to hop, we should think of ways to break up the gradient into parts that can hop separately with different sized hops. Non-stationary PEFs might do some of that.

What good is a 3-D PEF?

Although it's clear how to fit 3-D PEFs to data, I doubt the utility of it. When I see 3-D data, (t, x, y), I visualize it containing planes. A plane in 3-D looks like a line in both (t, x) and (t, y) space. It's more efficient to fit two planes each with a 2-D PEF $[a(t, x), a(t, y)]$ than with a single 3-D PEF $a(t, x, y)$. What kind of 3-D fields require 3-D PEFs? I don't know.

Segregating elastic waves

Naturally, we will attempt separation of pressure and shear waves with these multichannel methods. These methods compete, however with traditional scalar signal

methods based on the idea that S waves tend to have double the stepout of P, namely namely, $v^2 = (x/t)(dx/dt)$. This opens the door to fake recording channels.

Fake channels

Might multichannel technology be made more serviceable by faking extra channels as functions of the first? For example: $y_3(t,x) = y_1(t, x+\Delta x) - y_1(t, x-\Delta x)$.

I have a feeling oceanographers are well along in this area. Do a youtube search for "perpetual ocean" to see an awesome video. Physics involves curl and divergence, both of which motivate investigating the statistics of neighboring vector-field measurements. This video might look as though they have spatially dense measurements. Perhaps so, but we might also be seeing a well-crafted vector field made from coarser measurements.

DAS string

SEP's new DAS string introduces us to the uncomfortable notion of having one component of a vector quantity but not the other. It suggests a new form of missing data problem to solve. I'm not advocating we try solving it yet on our string, but I am ready to start discussing the problem.

Consistent wavelet polarities

Greg Beroza reminds us of repeating earthquakes. Especially small quakes may repeat with the same polarity. We should think about whether and how such phenomena can be best recognized.

CONCLUSION AND OPPORTUNITIES

In theory (hopefully) the nonstationary vector spectral factorization problem is solved. Its main application is segregating wave types in multicomponent data.

Of equal or greater interest is the solution methodology. Although introduced here in 1-D, it is generally applicable to higher dimensional data. It introduces nonstationary decon to many areas including regridding, missing data, and whitening inversion residuals.

- No need for synthetics. The simplicity of the nonstationary technology invites immediate experimentation with multidimensional field data.
- Robust applications (such as using the ℓ_1 norm) will become far more common.

- Sparse model estimation is an easy extension of the usual ℓ_2-norm regularization.
- The multidimensional filter is like a small molecule. It is built on statistics from a larger, differently shaped, region of data.
- New avenues arise for handling waveform variation over (shot-receiver) offset.
- Experimentation with ϵ will lead to deeper understanding of nonstationarity.
- Finite difference representation of differential equations (curl, divergence) might induce very weak non minimum phase that can be overcome via sparsity.

ACKNOWLEDGEMENT

I'd like to thank Enders Robinson for teaching me 55 years ago about stationary multichannel time series. Multicomponent seismograms were rare then (and imaging soon to be more fun). I thank Kaiwen Wang for showing my initial unitary transformation was totally wrong. I thank Joseph Jennings for thoughtful comments on early versions of this manuscript. He picked up numerous errors and identified several incomplete explanations. I'd also like to thank Mohamed Hadidi who helped me take a derivative with respect to a matrix. I also thank Carl Wunsch who brought the perpetual-ocean video to my attention.

REFERENCES

Claerbout, J., 2014, Geophysical image estimation by example: Lulu.com.

APPENDIX I: GRADIENT DERIVATION

I proposed the nonstationary problem (29) to Sergey Fomel who solved it. He solved it with powerful algebraic tools. The detailed algebra is found in earlier work by Fomel and Claerbout. Then I discovered that a simple step down the gradient led to the same updating of the filter. Though differing philosophically [(1) deviate from stationarity, and (2) step down the gradient], both methods update the filter with $\Delta\mathbf{a} = -\epsilon\, e\, \mathbf{d}$. Both philosophies obligate the practitioner to choose a suitable numerical value for ϵ. Finally, while preparing a lecture, I realized what you saw near the beginning of this paper that a simple approach almost devoid of mathematics leads to $\Delta\mathbf{a} = \pm\epsilon\, \mathbf{d}$ which amounts to a nonstationary ℓ_1-norm PEF. Below I derive algebraically the ℓ_2-norm gradient.

Start with any old PEF. We are going to improve it a tiny amount by considering just one new data value d_{n+1}. Call the old PEF $\bar{\mathbf{a}} = (1, \bar{a}_1, \bar{a}_2, \bar{a}_3, \cdots)$. The updated PEF $\mathbf{a}$ will be made by moving a small distance down the gradient (opposite polarity of the gradient).

Consider the regression:

$$\left[\begin{array}{c|ccc} d_{n+1} & d_n & d_{n-1} & d_{n-2} \\ \hline \lambda & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & \cdot \\ \cdot & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \lambda \end{array}\right] \left[\begin{array}{c} 1 \\ a_1 \\ a_2 \\ a_3 \end{array}\right] \quad \approx \quad \left[\begin{array}{c} 0 \\ \hline \lambda \\ \lambda\bar{a}_1 \\ \lambda\bar{a}_2 \\ \lambda\bar{a}_3 \end{array}\right] \tag{29}$$

The top block says we seek a PEF $\mathbf{a}$, that should improve fit of the newly arrived data value d_{n+1}. The lower block needs a large numerical value for λ to limit the amount of filter change. Define the fitting residual $\mathbf{r}$

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \left[\begin{array}{ccc} d_n & d_{n-1} & d_{n-2} \\ \hline \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{array}\right] \left[\begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array}\right] \quad - \quad \left[\begin{array}{c} -d_{n+1} \\ \hline \lambda\,\bar{a}_1 \\ \lambda\,\bar{a}_2 \\ \lambda\,\bar{a}_3 \end{array}\right] \tag{30}$$

Let

$$\mathbf{d} = \left[\begin{array}{c} d_n \\ d_{n-1} \\ d_{n-2} \end{array}\right] \quad , \quad \mathbf{a} = \left[\begin{array}{c} a_1 \\ a_2 \\ a_3 \end{array}\right]$$

With these definitions and $\mathbf{I}$ being an identity matrix the residual definition (30) is

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \left[\begin{array}{c} \mathbf{d}^T \\ \lambda\,\mathbf{I} \end{array}\right] \mathbf{a} \; - \; \left[\begin{array}{c} -d_{n+1} \\ \lambda\,\bar{\mathbf{a}} \end{array}\right] \tag{31}$$

We take the derivative of $\mathbf{r}^T\mathbf{r}$ to find the search direction.

$$\Delta\mathbf{a} \quad = \quad -\,(\text{some constant})\, \frac{\partial}{\partial \mathbf{a}^T}\bigg|_{\mathbf{a}=\bar{\mathbf{a}}} \mathbf{r}^T\mathbf{r} \tag{32}$$

Form the transpose of the residual (31), and then differentiate it by $\mathbf{a}^T$. (By $\mathbf{a}^T$ we mean the complex conjugate transpose of $\mathbf{a}$.)

$$\frac{\partial \mathbf{r}^T}{\partial \mathbf{a}^T} \quad = \quad \frac{\partial}{\partial \mathbf{a}^T} \left\{\mathbf{a}^T[\mathbf{d} \quad \lambda\mathbf{I}] \; - \; [-d_{n+1} \quad \lambda\bar{\mathbf{a}}]\right\} \quad = \quad [\mathbf{d} \quad \lambda\mathbf{I}] \tag{33}$$

Multiply that onto $\mathbf{r}$ from (31) keeping in mind that $\mathbf{d}^T\bar{\mathbf{a}}$ is a scalar and that the expression $(\mathbf{d}^T\bar{\mathbf{a}} + d_{n+1})$ is the prediction error e.

$$\frac{\partial \mathbf{r}^T}{\partial \mathbf{a}^T} \quad \mathbf{r} \quad = \quad [\mathbf{d} \quad \lambda\mathbf{I}] \left\{ \left[\begin{array}{c} \mathbf{d}^T \\ \lambda\,\mathbf{I} \end{array}\right] \mathbf{a} \; - \; \left[\begin{array}{c} -d_{n+1} \\ \lambda\,\bar{\mathbf{a}} \end{array}\right] \right\} \tag{34}$$

$$= \quad \mathbf{d}(\mathbf{d}^T\mathbf{a}) + \lambda^2\mathbf{a} + \mathbf{d}d_{n+1} - \lambda^2\bar{\mathbf{a}} \tag{35}$$

$$\frac{\partial \mathbf{r}^T}{\partial \mathbf{a}^T}\bigg|_{\mathbf{a}=\bar{\mathbf{a}}} \mathbf{r} \quad = \quad (\mathbf{d}^T\bar{\mathbf{a}} + d_{n+1})\mathbf{d} \tag{36}$$

$$\Delta\mathbf{a} \quad = \quad -\,\frac{1}{\lambda}\, e\, \mathbf{d} \tag{37}$$

Scale out the physical dimensions to see the filter update we've been using from the beginning of this paper.

APPENDIX II: RATFOR CODE (UNTESTED)

Compared with earlier pseudocode where the gradient is an unscaled adjoint, here the gradient has divided out the variances σ_e and σ_y. You may always scale gradient components by positive numbers.

```
# Non-stationary prediction error for vector signals in Ratfor/Fortran syntax
#
integer it, nt=1000, ia, na=10, gap=1, lambda=4000, ic, jc, nc=2
real y(nc,nt), e(nc,nt), aa(nc,nc,na), sige(nc), sigy(nc), eps
eps = 1./lambda
do ic=1,nc {
do jc=1,nc {
do ia=1,na {                    #   ia  =  the lag axis (formerly tau).
    aa(ic,jc,ia) = 0.
    }}}
do ic=1,nc {
    aa(ic,ic,1) = 1.            #  Identity matrix.
    }
do ic=1,nc {
do it=1,nt {
    e(ic,it) = 0.
    }}
read input y(nc,nt)             #  Read multichannel data.

do ic=1,nc {                    #  Initial variance estimates.
    sumsq=0
    do it=1,nt
        sumsq += y(ic,it)**2
    sigy(ic) = sqrt(sumsq/nt)
    sige(ic) = sigy(ic)
    }
#                               Here we go!  Happy streaming.  Wheee!
do it= na, nt {

    do ic=1,nc
        e(ic,it) = 0.
    do ia=1,na {                #  lag axis.
    do ic=1,nc {                #  Vector into a matrix of filters.
    do jc=1,nc {                #
        e(ic,it)     +=      aa(ic,jc,ia)         *   y(jc, it-ia+1)
        }}}
                                # Running variance.
    do ic=1,nc {     sigy(ic) = (1-eps)*sigy(ic) + eps*sqrt( y(ic,it)**2) }
    do ic=1,nc {     sige(ic) = (1-eps)*sige(ic) + eps*sqrt( e(ic,it)**2) }

    do ia=gap+1, na {           # adjoint = e * y'
    do ic=    1, nc {           #
    do jc=    1, nc {           #
        aa(ic,jc,ia) -= eps * (e(ic,it)/sige(ic)) * ( y(jc, it-ia+1) /sigy(jc))
        }}}
   }
```