Streaming nonstationary prediction error (I)

Jon Claerbout

ABSTRACT

A time-adaptive deconvolution filter is estimated and used in a streaming manner.

INTRODUCTION

Most of the time-series literature (perhaps all of its textbooks) presume data spectra are time invariant (a.k.a. stationary) while most applications involve spectra that change in time and space. Hence there is a strong need for nonstationary tools. Textbook theory (GIEE) tells us that PEF (prediction error filter) output is spectrally white. We design here a TV-PEF (time variable PEF) that pushes output towards a steady white even while the input spectrum varies. Statistical theory tells us we need IID residuals. In signal and image estimation practice the statistical term IID (Independent Identically Distributed) means that signals and images are white (the leading "I") and of uniform variance (the "ID"). Here deals only with the first "I". The approach here is "streaming," meaning that the entire data volume need not be kept in memory—it all flows through the box that I define here.

ALGORITHMS

The regression (1) is a textbook prediction-error regression with a weighting function w_t diminishing from the present time t to the past. We define one step of streaming as entering a new data row at the bottom while dropping an old one at the top. The previous best-fitting PEF now needs revision from $\mathbf{a}$ to $\mathbf{a} + \Delta\mathbf{a}$. We set out to estimate $\Delta\mathbf{a}$. Rather than expect "perfection", at each moment in time we take just one step in the direction of perfection, and then move on to the next d_t.

$$\mathbf{0} \approx \begin{bmatrix} r_{t-4} \\ r_{t-3} \\ r_{t-2} \\ r_{t-1} \\ r_t \end{bmatrix} = \begin{bmatrix} w_4 & \cdot & \cdot & \cdot & \cdot \\ \cdot & w_3 & \cdot & \cdot & \cdot \\ \cdot & \cdot & w_2 & \cdot & \cdot \\ \cdot & \cdot & \cdot & w_1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & w_0 \end{bmatrix} \begin{bmatrix} d_{t-4} & d_{t-5} & d_{t-6} \\ d_{t-3} & d_{t-4} & d_{t-5} \\ d_{t-2} & d_{t-3} & d_{t-4} \\ d_{t-1} & d_{t-2} & d_{t-3} \\ d_t & d_{t-1} & d_{t-2} \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} = \mathbf{WDa} \tag{1}$$

Application types

1. A universal application is whitening residuals while fitting data to a model. The residual of that modeling is injected here as the data in regression (1).

2. Another application is interpolating data d_t having occasional missing values. In the bottom row of (1) we would assert $r_t = 0$ and then find the missing value by $d_t = -a_1 d_{t-1} - a_2 d_{t-2}$. (To understand why this is a good estimate you could read in Chapter 7, Multidimensional Autoregression, in GIEE.)

3. Another application is synthesizing random data of a given spectrum. A random number sequence n_t is put in the left column in (1) and a synthetic (or modeled) data set s_t associated with the data matrix. The bottom equation in (1) now defines the synthesized data s_t recursively by $s_t = n_t - a_1 s_{t-1} - a_2 s_{t-2}$. More interesting than random numbers could be isolated impulses. Then the output at each spike would be the impulse response of the inverse PEF.

Updating

At every value of t the regression (1) has a different solution for $\mathbf{a}$. Upon a clock tick a new data value d_t arrives, older data values feel a weaker weight, and new residuals $\mathbf{r}$ need be computed. To find the new PEF $\mathbf{a}$ starting from the old one we use the textbook story of steepest descent. The gradient $\Delta\mathbf{a}$ is the new residual dumped into the adjoint; and the Newton method tells us the scale α to apply to the gradient for the update $\mathbf{a} \leftarrow \mathbf{a} + \alpha\Delta\mathbf{a}$.

$$\Delta\mathbf{a} = \mathbf{D}^T\mathbf{W}^T\mathbf{r} \qquad \text{except that } \Delta\mathbf{a}_0 = 0 \tag{2}$$

$$\Delta\mathbf{r} = \mathbf{W}\mathbf{D}\Delta\mathbf{a} \tag{3}$$

$$\alpha = -\frac{\mathbf{r}\cdot\Delta\mathbf{r}}{\Delta\mathbf{r}\cdot\Delta\mathbf{r}} \tag{4}$$

Would more iterations be helpful? Perhaps the opposite. Maybe we should scale down α to avoid pushing hard the filter on the basis of too little data.

For larger data gaps, two streams could run, one forwards another backwards. Gap values would average the two predictions each weighted by its nearness to known data.

The Clapp summer group code stores the filter $\mathbf{a}$ on a coarse mesh. Here we find no need to store the filter. As soon as it is determined, it is used and then abandoned. As for computer speed, this algorithm is recursive only in the outer loop.

PRELIMINARY 1-D RESULTS

Coding the first of the three "Application types," blind deconvolution with nonstationary PEFs, took me all day Saturday for 1-D and all Sunday for 2-D. Resulting

codes are short and sweet while the results are fascinating. In Figure 1 we see a sequence of damped exponential functions $\text{step}(t - t_{\text{delay}})e^{-\alpha t} \sin \beta t$. The sequence

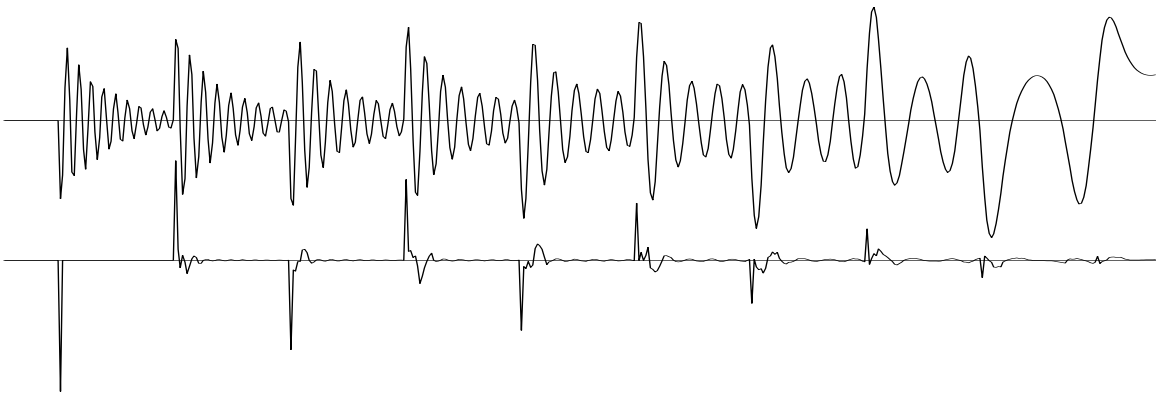

Figure 1: A synthetic signal (top), and its nonstationary prediction error (bottom). Seismologist love it when complicated signals are reduced to a sequence of impulses. [**ER**]

begins with high frequency functions. Later the high frequency parts are dissipated away as you expect in an absorbent earth. In the beginning the damped pulses hardly overlap, but later on they overlap a lot. A single three term PEF, such as $(1, a_1, a_2)$, is also a finite difference equation for a damped sinusoid so it is no surprise that the first damped sinusoid is collapsed back to a pulse at its onset. Seismologists love it when that happens. But the quality of the zeros degrades towards the end of the time axis. That's where the exponentials are lower frequency and overlap significantly. The PEF has a harder time getting rid of many frequencies at the same time. The PEF could do better if I gave it more coefficients (longer wavelet), but that would require more history to estimate. And, more cost with more iterations.

Code

First examine central part of the 1-D code. I explain it on a video. (A long Saturday but only 44 lines of executable code.(!))

```
do ia=1,na { aa(ia) = 0.}
aa(1)=1.;
do ir = 1, n1 {
        do iter=1,niter {
#         MATH b = D a  (a is a PEF.  b is a box of local residuals.  D is convolution with data.)
               do ib = 1, nb { rb(ib) = 0.}     # rb is a box for the local residual.
               do ib = 1, min0(nb,ir)      {
               do ia = 1, min0(na,ir-ib+1) {
                                rb(ib) += dd(ir-ib-ia+2) * aa(ia)
                                } }
               rr(ir) = rb(1)     # Grab result!
#         MATH da = D' b    (D' is adjoint D)
               do ia = 1, na { da(ia) = 0.}
               do ib = 1, min0(nb,ir)      {
               do ia = 1, min0(na,ir-ib+1) {
```

```
                                da(ia) += dd(ir-ib-ia+2) * rb(ib)
                                } }
                da(1) = 0.              # Constraint
#           MATH db = D da    (db is residual perturbation)
                do ib = 1, nb { dr(ib) = 0.}
                do ib = 1, min0(nb,ir)    {
                do ia = 1, min0(na,ir-ib+1) {
                                dr(ib) += dd(ir-ib-ia+2) * da(ia)
                                } }
#           MATH steepest descent
                nbmax = min0(nb,ir-na)
                nbmax =      nb
                top=0.;  bot=1.e-20
                do ib = 1, nbmax {
                        top += rb(ib) * dr(ib) * (nb-ib+1)     # linear taper
                        bot += dr(ib) * dr(ib) * (nb-ib+1)
                        }
                alpha = -  top/bot
                do ia = 1, na {
                        aa(ia) += alpha * da(ia)
                        }
                }
        }
```

TWO PHYSICAL DIMENSIONS

Let us take a look at a fragment of the 2-D code. It is pretty much like the 1-D code, but for two differences. As you see in GIEE Chapter 4, the 2-D PEF has its impulse along one side with zero values constrained either above or below the impulse. The 2-D calculation of residuals is an obvious extension of the 1-D code, and likewise the adjoint. Here's the fragment. It's the calculation of the residual.

```
do ia1=1,na1 {
do ia2=1,na2 {
        aa(ia1,ia2) = 0.
        }}
           lag = (na1+1)/2
        aa(lag,1) = 1.          # Put spike on side of filter.
do ir2 = 1, n2 {
do ir1 = lag, n1 {
        do iter=1,niter {
                do ib1=1, nb1{
                do ib2=1, nb2{
                        rb(ib1,ib2) = 0.
                        }}
                do ib1=1, min0( nb1, ir1) {
                do ib2=1, min0( nb2, ir2) {
                        do ia1 = 1, min0( na1, ir1-ib1+1) {
                        do ia2 = 1, min0( na2, ir2-ib2+1) {
                                rb(ib1,ib2) += dd(ir1-ib1-ia1+2,ir2-ib2-ia2+2) * aa(ia1,ia2)
                                } }
                        } }
                rr(ir1-lag+1,ir2) = rb(1,1)    # Here goes the output.
                etc.
                etc.
                do ia1= 1, lag {
                        da( ia1 ,1) = 0.               # constrain the 1.0 and the 0.0s
                        }
                etc.
```

Ignoring parameter fetching and declarations, the 2-D code is a mere 67 lines which resembles the 1-D code except with 2-D convolutions like the above fragment.

Figure 2 shows a test case. This example was previously demonstrated by the Clapp summer group with alternate coding strategies, and by Bob himself with pretty much this strategy. He and I should get together to make comparisons because comparisons are unfair without comparable parameters. Bob makes the point that after dropping off the bottom of the time axis and going to the top point on the space axis, you don't have the best guess for the filter (because early times and late times have different statistics). One option would be to save the top filter from the previous x. A simpler option is to run a few more steepest descent iterations for the topmost filters.

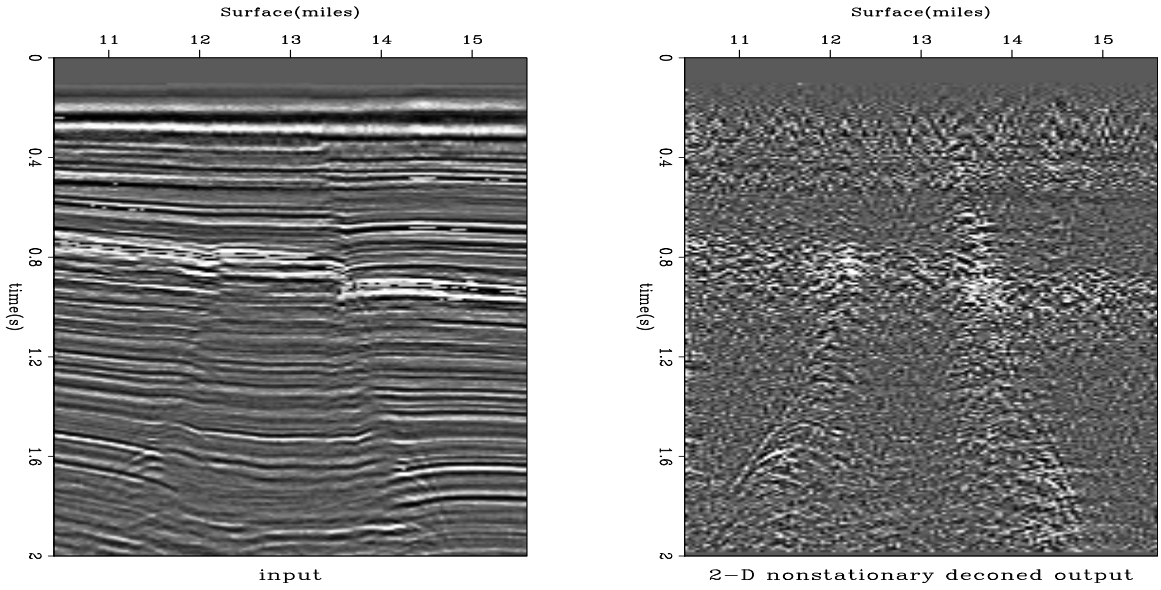

Figure 2: Data (left) is dominated by rather flat horizontal reflectors obscuring fault-plane reflections. Nonstationary 2-D PEF brings them out. We should discuss parameter choices leading to plots like this one. What you see here is my first wild guess. na1=5 na2=2 nb1=10 nb2=5 niter=4 [**ER**]

STABILITY

Stability is not as bad an issue as it sounds. Basically, a prediction-error filter is able to predict a growing exponential. If your data looks like a growing exponential at the edge of any gap (or boundary) the predicted data will grow exponentially with distance into the hole. It is easy enough to put the exponential growth under a damped exponential. A single parameter must be chosen. This parameter has little effect in a small hole, so its value doesn't come into play until you start dealing with large holes. Simply start diminishing the PEF coefficients with every step into the hole. For any small $\epsilon > 0$ in 2-D, alter the estimated PEF by $a_{i,j} \leftarrow a_{i,j} \times (1-\epsilon)^{|i|+|j|}$

where the unit spike is at $(i, j) = (0, 0)$. A similar issue and solution arises with the third application type.

Reality is that real data is noisy enough that we are unlikely to run into the highly predictable situations that bring on such growth. We might see it though extending a data plane where the last seismogram was anomalously larger than earlier ones.