Fitting while whitening nonstationary residuals

Jon Claerbout

ABSTRACT

While fitting a model to given data, I simultaneously whiten nonstationary residuals in both data space and model space. Last year we learned, in the presence of nonstationary noise in multidimensional space, how to solve three classes of problems: (1) deconvolution, (2) missing data, and (3) simulating data with the nonstationary spectrum of given data. This year I extend that class of problems to a general model $\mathbf{m}$ building for data fitting, $\Delta\mathbf{m} = \mathbf{F}^T\mathbf{A}^T\mathbf{A}(\mathbf{Fm} - \mathbf{d})$. In a space of any dimension we need only store $\mathbf{A}$, a Prediction Error Filter (PEF) in the zone of the transition from the filtered to the unfiltered. I propose nine test cases and projects.

INTRODUCTION

Multidimensional PEFs are powerful data analysis tools that can deal effectively with spatially aliased data. Most of the time-series literature (perhaps all of its textbooks) presume data spectra are time invariant (*aka* stationary) while most applications involve spectra that change in time and space. Hence there is a strong need for nonstationary tools. Textbook theory such as my book GIEE[1] tells us that PEF output tends to spectrally white.

We design here a time variable PEF (TV-PEF) that pushes output towards a steady whiteness even while the input spectrum varies. Statistical theory tells us we need Independent Identically Distributed (IID) residuals. In signal and image estimation practice the statistical term "IID" means that signals and images are spectrally white (the leading "I") and of uniform variance (the "ID"). Here I deal 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. Whether your entire process allows streaming naturally depends on whether your $\mathbf{F}$ operator in $\mathbf{0} \approx \mathbf{Fm} - \mathbf{d}$ allows it.

We also overcome a fundamental conundrum. Inverse theory, invented by mathematicians, says it will guide us to the "best" solution, but first we must supply a good deal of statistical information about the "best" solution. Whoa! Additionally, we must supply a good deal of statistical information about the misfit of theory and experiment. Hold your horses! If we haven't seen a solution yet, how can we know the statistics of the misfit of models to data?

[1] `http://sep.stanford.edu/sep/prof/gee/book-sep.pdf` page 182

We're pragmatic people. Let us not be so fussy. We can always begin by guessing that the data misfit statistics will resemble the data statistics. Since we study seismic image estimation it seems reasonable to start with the assumption that the data variance, decreasing as it does with t^{-2}, matches the data residual variance. That's for the time domain. Likewise for space. In this paper we work only with multidimensional filtering, which amounts to weighting in the multidimensional frequency domain.

In principle, PEFs whiten signals (GIEE). We are just starting out. We'd like to know the PEF that whitens the ultimate fitting residuals that we don't have yet. So, let us change the PEF at every iteration. By whitening our current residuals we whiten our final residuals. This is more proper than guessing that the final residual spectrum is the same as the data spectrum! (Theory states that iteration-variable weighting harms some solvers, while others are immune.) What happens to people who use suboptimal weights and PEFs? They do not use their data "efficiently" (a statistical term). We are ready to go, wondering if we might consistently outperform our forebears in iteration count as well as solution quality. Even if not, we remain enthusiastic. The tool here handles both **nonstationarity** and **spatial aliasing**, massive issues in seismic imaging.

BEGIN IN ONE DIMENSION

Begin with 1.5 pages of review of work I did last year with Sergey. Suppose we have a PEF that represents all previous moments in time. Call it $\bar{\mathbf{a}} = (1, \bar{a}_1, \bar{a}_2, \bar{a}_3, \cdots)$. Say that $\bar{\mathbf{a}}$ represents data values $(d_1, d_2, d_3, \cdots, d_{98})$. We seek to define the $\mathbf{a}$ that represents that data with an appended data value d_{99}.

Consider the regression:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \approx \left[\begin{array}{cccc} d_{99} & d_{98} & d_{97} & d_{96} \\ \hline \gamma & \cdot & \cdot & \cdot \\ \cdot & \gamma & \cdot & \cdot \\ \cdot & \cdot & \gamma & \cdot \\ \cdot & \cdot & \cdot & \gamma \end{array}\right] \begin{bmatrix} 1 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} - \gamma \left[\begin{array}{c} 0 \\ \hline 1 \\ \bar{a}_1 \\ \bar{a}_2 \\ \bar{a}_3 \end{array}\right] \tag{1}$$

The top row says we are trying to fit a new data point d_{99}. The bottom block says the new PEF $\mathbf{a}$ should be pretty similar to the PEF that fit earlier data, $\bar{\mathbf{a}}$. The parameter γ should be big enough that the new data point d_{99} does not change $\mathbf{a}$ very much. Rewrite equation (1) as

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \approx \begin{bmatrix} d_n & d_{n-1} & d_{n-2} \\ \gamma & 0 & 0 \\ 0 & \gamma & 0 \\ 0 & 0 & \gamma \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} - \begin{bmatrix} -d_{n+1} \\ \gamma\, \bar{a}_1 \\ \gamma\, \bar{a}_2 \\ \gamma\, \bar{a}_3 \end{bmatrix} \tag{2}$$

or, in a shortened block-matrix notation, we have the residual to minimize

$$\mathbf{0} \quad \approx \quad \mathbf{r} \quad = \quad \begin{bmatrix} \mathbf{d}^T \\ \gamma\,\mathbf{I} \end{bmatrix} \mathbf{a} \;-\; \begin{bmatrix} -d_{n+1} \\ \gamma\,\bar{\mathbf{a}} \end{bmatrix}, \tag{3}$$

where $\mathbf{I}$ is the identity matrix and

$$\mathbf{d} = \begin{bmatrix} d_n \\ d_{n-1} \\ d_{n-2} \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix},$$

For decades Bernard "Bernie" Widrow (Wikipedia) attacked problems of this nature by defining a quadratic form and finding its gradient. Then he repeatedly made small steps down the gradient (not up). How big are the small steps? Experience teaches. The quadratic form is $\mathbf{r}^T\mathbf{r}$. We take its derivative to find the search direction.

$$\Delta\mathbf{a} \quad = \quad -\,(\text{some constant})\; \frac{\partial}{\partial \mathbf{a}^T}\bigg|_{\mathbf{a}=\bar{\mathbf{a}}} \mathbf{r}^T\mathbf{r} \tag{4}$$

Form the transpose of the residual (3) and then differentiate by $\mathbf{a}^T$. (By $\mathbf{a}^T$ we mean the complex conjugate transpose of $\mathbf{a}$.)

$$\frac{\partial \mathbf{r}^T}{\partial \mathbf{a}^T} \quad = \quad \frac{\partial}{\partial \mathbf{a}^T}\; \Big\{\mathbf{a}^T[\mathbf{d} \quad \gamma\mathbf{I}] \;-\; [-d_{n+1} \quad \gamma\bar{\mathbf{a}}]\Big\} \quad = \quad [\mathbf{d} \quad \gamma\mathbf{I}] \tag{5}$$

and multiply that onto $\mathbf{r}$ from (3) keeping in mind that $\mathbf{d}^T\bar{\mathbf{a}}$ is a scalar.

$$\Delta\mathbf{a} \quad \propto \quad \frac{\partial \mathbf{r}^T}{\partial \mathbf{a}^T}\;\mathbf{r} \quad = \quad [\mathbf{d} \quad \gamma\mathbf{I}] \left\{ \begin{bmatrix} \mathbf{d}^T \\ \gamma\,\mathbf{I} \end{bmatrix} \mathbf{a} \;-\; \begin{bmatrix} -d_{n+1} \\ \gamma\,\bar{\mathbf{a}} \end{bmatrix} \right\} \tag{6}$$

$$= \quad \mathbf{d}(\mathbf{d}^T\mathbf{a}) + \gamma^2\mathbf{a} + \mathbf{d}d_{n+1} - \gamma^2\bar{\mathbf{a}} \tag{7}$$

$$\Delta\mathbf{a} \quad \propto \quad \frac{\partial \mathbf{r}^T}{\partial \mathbf{a}^T}\bigg|_{\mathbf{a}=\bar{\mathbf{a}}}\;\mathbf{r} \quad = \quad (\mathbf{d}^T\bar{\mathbf{a}} + d_{n+1})\;\mathbf{d} \tag{8}$$

It is certainly surprising that the analytic solution to the regression (1) computationally amounts to a single step of the optimization strategy (8), a strategy so crude as to be absent from textbooks. Yet that is so (REFER TO SEP REPORT)[2] Experimentalists will first notice that (1) demands we supply a not-given constant γ while (8) and its coming equivalent (9) demands a not-given constant λ.

Now we recast (8) so as to best understand how to choose the not-given step size. The expression $(\mathbf{d}^T\bar{\mathbf{a}} + d_{n+1})$ in (8) is the prediction error. The prediction filter $\mathbf{a}$ takes data to predicted data, so it has no physical units. The parameter we define for the aggressiveness of PEF adaptation will be unit free. The gradient (8) has units of data squared so it needs a normalizing factor of the same units such as a local data variance $\hat{\sigma}_d^2$. Now we choose a dimensionless parameter λ that we will learn from

[2]http://sep.stanford.edu/sep/jon/streamingSergey.pdf

experience. I put λ in the denominator so the larger the value of λ the slower the adaptation rate. Thus λ represents something like the size of the statistical averaging region measured in pixels. I would begin experimentation with $\lambda = 100$.

$$\Delta\mathbf{a} \quad = \quad \mathbf{a} - \bar{\mathbf{a}} \quad = \quad -\frac{1}{\lambda}\left(\frac{\text{prediction error}}{\hat{\sigma}_d^2}\right)\mathbf{d} \tag{9}$$

We likely wish to define local variance $\hat{\sigma}_d^2$ with the same averaging region that we are defining the PEFs, so

$$\hat{\sigma}_{d_t^2} \quad = \quad \frac{\lambda - 1}{\lambda}\,\hat{\sigma}_{d_{t-1}^2} \;+\; \frac{1}{\lambda}\,d_t^2 \tag{10}$$

Perhaps the divisor in equation (9) should scale by $\sigma_r\sigma_d$ instead of σ_d^2. Experience will show.

It is almost obvious how and why Equation (9) works. It adds a little piece of data $\mathbf{d}$ to the filter $\Delta\mathbf{a}$. That same little piece of data is going into the filter and $\mathbf{d}\cdot\mathbf{d} > 0$. Whether that piece should be scaled positively or negatively depends on the polarity of the prediction error (which depends on d_{n+1}). Stunningly simple, is it not? And the cost is near nothing.

APPLYING THE ADJOINT OF A STREAMING FILTER

Those of us with a long history of filtering think of a filter adjoint as running the filter backwards. That view arises with recursive filters whose adjoint must indeed run backwards. With nonrecursive filters, such as prediction error, there is a more basic view. In a (nonrecursive) linear operator program, the inputs and outputs can be exchanged to produce the adjoint. For example the pseudocode below applies a filter "a" to data "d" to get a residual "r".

```
do it = na, nt {
        do ia = 1, na {
                if operator itself
                        y(it) += x(it-ia+1) × a(ia)        # one output y(t) pulls
                if adjoint
                        x(it-ia+1) += y(it) × a(ia)        # one input y(t) pushes
                }
        }
```

Observe the time axis runs forward for both the operator and its adjoint $\mathbf{A}^T$. To

update a model requires both a PEF $\mathbf{A}$ and its adjoint. Define

$$\begin{aligned} \mathbf{r} &= (\mathbf{Fm} - \mathbf{d}) && (11)\\ \mathbf{q} &= \mathbf{A}(\mathbf{Fm} - \mathbf{d}) && (12)\\ \mathbf{s} &= \mathbf{A}^T\mathbf{A}(\mathbf{Fm} - \mathbf{d}) && (13)\\ \Delta\mathbf{m} &= \mathbf{F}^T\mathbf{A}^T\mathbf{A}(\mathbf{Fm} - \mathbf{d}) && (14) \end{aligned}$$

You take care of $\mathbf{F}$ and $\mathbf{r}$. I next give you pseudocode for $\mathbf{s} = \mathbf{A}^T\mathbf{Ar}$.

```
do it = na, nt {
        Using equation (9), update filter a(ia=1,na)
        do ia = 1, na {
                q(it) += r(it-ia+1) × a(ia)          # pull
                }
        do ia = 1, na {
                s(it-ia+1) += q(it) × a(ia)          # push
                }
        Move a to abar.
        }
```

Notice that the program also works when the time axis is run backwards. In two dimensions, either or both the axes may be run backwards. Flipping axes flips the region in which statistics are gathered.

AVERAGING IN TIME AND SPACE

A streaming 1-D prediction filter is an average of earlier prediction filters, however these earlier filters need not be all saved in memory. Since they vary smoothly we may simply use the most recent one. Take the $\bar{\mathbf{a}}$ of one dimension. In two dimensions, for example time and space, it becomes some average of its previous value on each of those two axes.

$$\bar{\mathbf{a}}(t,x) \quad = \quad \cos^2\theta\ \mathbf{a}(t-\Delta t, x) \ + \ \sin^2\theta\ \mathbf{a}(t, x-\Delta x) \tag{15}$$

where θ is an adjustable parameter for the user to specify the shape of the region for gathering statistics. With equation (15), equation (9) becomes

$$\mathbf{a}(t,x) \quad = \quad \bar{\mathbf{a}}(t,x) \quad - \quad \frac{1}{\lambda}\left(\frac{\text{prediction error}}{\hat{\sigma}_d^2}\right)\mathbf{d} \tag{16}$$

Sweeping across the x axis for all t requires in memory all PEFs at $x-\Delta x$, but only the one PEF at $t-\Delta t$. In 3-D it looks like we will need a plane of PEFs. In higher dimensional spaces we need store PEFs only in the zone of the transition from the

filtered to the unfiltered. Thus in 5-D we need to store a 4-D volume of PEFs. Don't let that trouble you though. Since the PEFs are all smoothly variable they could be linearly interpolated from a sparse mesh. PEFs on the previous trace $\mathbf{a}(t, x - \Delta x)$ might be smoothed symmetically on the time axis so the region of averaging expands from a quadrant to a halfspace.

PEFS FOR BOTH FITTING AND STYLING

Having PEF $\mathbf{A}$ on the regularization and PEF $\mathbf{B}$ on the fitting, the gradient is

$$\Delta\mathbf{m} = \mathbf{F}^T\mathbf{B}^T\mathbf{B}\mathbf{r} + \epsilon^2\mathbf{A}^T\mathbf{A}\mathbf{m} \tag{17}$$

which may be coded as we did equations (11)-(14). Naturally, the PEF $\mathbf{B}$ is applicable only on those data-space axes that are regularly sampled such as along a marine cable.

We have ignored preconditioning. It's important for covering large gaps such as at cable ends. But in most applications we have more modest goals such as data sampling irregularities and gaps the size of streamer separations. Furthermore, the speed of this method might render preconditioning irrelevant even on larger gaps.

Big stoppers for conventional analysis are two: (1) nonstationarity (2) spatial aliasing. Our PEFs can handle both.

CONCLUSIONS AND FUTURE WORK

The theory and programming sketch above seems fairly complete and ready to run. Some of us may use `cgstep` while new students use `sepsolve`. Here are some preliminary tests, some of them pathfinders of textbook value.

1. **Extend a shot gather:** We should see conflicting dips extended appropriately off the cable ends. We could always try preconditioning (polynomial division), but because the new methods are ultra fast they alone might compensate for lack of preconditioning?

2. **Velocity transform:** It's always fun to transform (t, x) to (τ, s). Do we automatically get sharp resolution? Can we obtain more sparsity in model space simply by introducing softclip? This is really promising and should be easy to test. Is it relevant or irrelevant to estimate off-end traces in data space?

3. **Streaming interpolation while stacking** Could we regularize and interpolate data, then add it into the final image without ever storing the interpolated data? Start by assuming one of every three traces is missing.

4. **Flip flop shooting** In multistreamer seismic data acquisition left- and rightside guns fire alternating pops. How is this paper relevant?

5. **Madagascar:** Should be a pleasing demonstration of nonstationarity. There are a few spikes in the interior, and many around the periphery but code exists to mask out those data values.

6. **Vesuvius:** Are we correct to expect different PEFs on different sides of the mountain? Textbook method lacks weights, but they are easy to include. Textbook also coarse binned the given high density data partly to reduce noise but mainly to speed computation. Is coarse binning still necessary?

7. **Galilee:** To avoid all the noise we could use the solution from the book to make noise-free data. On the real data, large noise is the main issue. It would be delightful should we discover replacing the data residual by its softclip solves the problem of nonstationary model with bursty noise.

8. **Flipping axes:** Running a time or space axis backwards flips the quadrant in which statistics are gathered. What are the disadvantages and possible advantages? When should we be running both ways?

9. **Deep learning:** Start from the assumption that one of the above projects works well. By testing a wide range of shapes and sizes (λ and θ) we should gain insights about the appropriate choice. Nonstationary filters adapt to changing changing spectra, but they don't know how slow (big size) to adapt. Can we quantify it, teaching them to learn? See Bernard Widrow at Wikipedia.

SIGNAL GRIDDING

Old thoughts

A universal problem in applied geophysics is signals (time functions) on irregular (x, y)-space axes. Denote the signals $\mathbf{d}_t(x, y)$. With $\mathbf{L}$ being linear interpolation and $\mathbf{A}$ being a regularizer, the gridding problem might be posed as a collection of many 2-D regressions, each with the same operator.

$$\begin{bmatrix} \mathbf{L} \\ \mathbf{A} \end{bmatrix} [\cdots \mathbf{m}_t \mathbf{m}_{t+1} \mathbf{m}_{t+2} \cdots] \quad \approx \quad \begin{bmatrix} \cdots \mathbf{d}_t \mathbf{d}_{t+1} \mathbf{d}_{t+2} \cdots \\ \cdots \mathbf{0} \cdots \end{bmatrix} \tag{18}$$

Write the collection of desired solutions as $\mathbf{M} = [\cdots \mathbf{m}_t \mathbf{m}_{t+1} \mathbf{m}_{t+2} \cdots]$ and likewise for $\mathbf{D}$. The analytic solution to the regression (18) is $\mathbf{M} = (\mathbf{L}^T\mathbf{L} + \epsilon^2 \mathbf{A}^T\mathbf{A})^{-1}\mathbf{L}^T\mathbf{D}$. Iterative solvers easily give a solution for any one of the $\mathbf{m}_t$, but we had no sensible way to come up with the inverse of the 2-D operator $(\mathbf{L}^T\mathbf{L} + \epsilon^2 \mathbf{A}^T\mathbf{A})^{-1}$. Given it, we would easily have $\mathbf{m}_t$ for all t, though in principle it's a full, dense matrix, so most likely would be approximated by a low order product of $\mathbf{L}$'s and $\mathbf{A}$'s which starts to look like solving the original regression at each t.

New thoughts

Let us try another approach. Define $\mathbf{m} = [\cdots \mathbf{m}_t \mathbf{m}_{t+1} \mathbf{m}_{t+2} \cdots]$ as a 3-D (t, x, y) space. Like equations (17) and (11)-(14) we may deduce PEFs $\mathbf{B}$ and $\mathbf{A}$ and the gradient $\Delta\mathbf{m}$

$$\Delta\mathbf{m} = \mathbf{L}^T\mathbf{B}^T\mathbf{B}(\mathbf{L}\mathbf{m} - \mathbf{d}) + \epsilon^2\mathbf{A}^T\mathbf{A}\mathbf{m} \tag{19}$$

Naturally, the PEF $\mathbf{B}$ is nontrivial only on data-space axes that are regularly sampled. Equation (19) is fundamentally better than (18) because we do not impose the PEFs but find them. Additionally, in (19) all the operators are local, suitable for streaming. Admittedly, we are iterating 2-D operators over the entire t-space. We recall the cost of finding PEFs is merely double the cost of applying them. But surely, this cannot fill big data gaps in (x, y) without iterating (19). I would first try $\epsilon = 1$.

As for the time axis, in principle we handle it with an infinitesimal adaptation rate, though in practice we'd find some other way to the limit. I won't understand this until I code it.