Adaptive prediction error filters

Rustam Akhmadiev

April 16, 2018

Abstract

The theory and the applications of prediction error filters are very well known. Most of the time, however, they are built to be stationary (not varying in space and time) and therefore, they carry some global statistical information of the signals. Hence, these filters cannot be expected to be optimal in the nonstationary environment.

This paper addresses the problem of designing nonstationary prediction error filter (PEF). Gradient adaptive lattice and recursive least-squares filters are used. Their one- and two-dimensional applications give justifications to their advantageous behavior compared with conventional stationary PEFs.

Introduction

Signals recorded on seismic records are as a rule not stationary. It is well known that the frequency content of the seismic waves change as they propagate in the subsurface due to the anelastic attenuation and scattering. At the same time recorded data is nonstationary in space since different events have different slopes that change with time as well. That is why it is natural to try to find the filters that will adapt to the local characteristics of the signals.

Prediction error filters arise in the theory of linear prediction, that deals with the problem of estimating the future samples of the signal using linear combination of its previous samples weighted by sought-for filter coefficients. The error of this prediction turns out to be white (not correlated), while the filter itself will have the inverse spectrum of the signal (that allows estimation of the original signal). Consequently, one of the most common application of these filters is deconvolution – the process that tries to widen the spectrum by removing the convolutional effects with the bandlimited signal. Having the information of the original signals, these filters can also be used for the multidimensional interpolation.

There are different ways of finding PEF all based on minimizing the norm of the prediction error. Probably, the very first successful way of solving this problem is based on an efficient way of solving normal equations (finding inverse of a Toeplitz matrix) using Levinson-Durbin algorithm (FGDP). This problem can also be approached using optimization theory tools like methods of steepest descent, conjugate gradient, etc (GIEE). While being very efficient and stable, the latter one, however, requires quite a lot of effort to be implemented in the nonstationary environment (JOES REPORT). On the other hand, it turns out that the methods based on the original Levinson-Durbin recursion are very fast, efficient and easily adjusted to account for nonstationarity.

One of these filters considered here is called lattice predictor filter and arises from the Levinson-Durbin algorithm itself. It is well studied in the electrical engineering society, it can be easily extended to be adaptive and it is known to be the most efficient structure to give orthogonal (not correlated) forward and backward prediction errors.

Another popular adaptive algorithm is called recursive least squares (RLS) that essentially is the way of adaptively finding inverse correlation matrix (coming from the normal equations) as the data streams through the filter.

Lattice filter

Lattice filter can be viewed from the point of view of Levinson-Durbin recursion. It is well known that the filter coefficients are updated in a following way:

$$\mathbf{a}_m = \begin{bmatrix} \mathbf{a}_{m-1} \\ 0 \end{bmatrix} + K_m \begin{bmatrix} 0 \\ \mathbf{a}^B_{m-1} \end{bmatrix} \tag{1}$$

where $\mathbf{a}_{m-1}$ - prediction-error filter of $(m-1)$-order and size (m) on the previous iteration, $\mathbf{a}^B_{m-1}$ - the same coefficients in the reverse order K_m - reflection coefficients, $\mathbf{a}_m$ – updated prediction-error filter of (m)-order and size $(m+1)$. Let's consider the action of the m-order PEF to the input $\mathbf{u}_{m+1} = [u(n), u(n-1), \dots, u(n-m)]$ of size $(m+1)$ and with the corresponding lags equal to $[0, 1, \dots, m]$. This produces m-order forward prediction error

$$f_m(n) = \mathbf{a}_m^T \mathbf{u}_{m+1}(n)$$

Looking at the terms in the equation 1 separately:

$$\begin{bmatrix}\mathbf{a}_{m-1}^T & 0\end{bmatrix} \mathbf{u}_{m+1}(n) = \begin{bmatrix}\mathbf{a}_{m-1}^T & 0\end{bmatrix} \begin{bmatrix}\mathbf{u}_m(n) \\ u(n-m)\end{bmatrix} = \mathbf{a}_{m-1}^T \mathbf{u}_m(n) = f_{m-1}(n)$$

$$\begin{bmatrix}0 & \mathbf{a}_{m-1}^{BT}\end{bmatrix} \mathbf{u}_{m+1}(n) = \begin{bmatrix}0 & \mathbf{a}_{m-1}^{BT}\end{bmatrix} \begin{bmatrix}u(n) \\ \mathbf{u}_m(n-1)\end{bmatrix} = \mathbf{a}_{m-1}^{BT} \mathbf{u}_m(n-1) = b_{m-1}(n-1)$$

So to estimate the forward prediction error $f_m(n)$ (where m represents the filter length $(m+1)$) at time n we need forward prediction error $f_{m-1}(n)$ at time n corresponding to the lower $(m-1)$-order filter and backward prediction error $b_{m-1}(n-1)$ at the previous sample $(n-1)$:

$$\begin{aligned} f_m(n) &= f_{m-1}(n) + K_m b_{m-1}(n-1) \\ b_m(n) &= b_{m-1}(n-1) + K_m f_{m-1}(n) \end{aligned} \tag{2}$$

Now the prediction filter can be represented as Figure **??**

In order for this procedure and the resulting filter to be stable (and minimum-phase) the prediction-error power should be decreasing or at least stay the same. Looking at the update equations for the prediction-error of the Levinson recursion (YILMAZ), it easy to see that this requires the reflection coefficients $|K_i| \leq 1$ at every iteration $i = 1, \dots, m$.

To find the coefficients K_m we should find the minimum of objective function:

$$J_m = ||\mathbf{f}_m(n)||^2 + ||\mathbf{b}_m(n)||^2$$

where $\mathbf{f}_m(n)$ and $\mathbf{b}_m(n)$ are now the vectors containing the forward and backward prediction errors of m-order up to n-th sample.

$$\begin{aligned} J_m &= ||\mathbf{f}_m(n)||^2 + ||\mathbf{b}_m(n)||^2 = ||\mathbf{f}_{m-1}(n) + K_m \mathbf{b}_{m-1}(n-1)||^2 + ||\mathbf{b}_{m-1}(n-1) + K_m \mathbf{f}_{m-1}(n)||^2 \\ &= (||\mathbf{f}_{m-1}(n)||^2 + ||\mathbf{b}_{m-1}(n)||^2)(1 + K_m^2) + 4K_m \mathbf{b}_{m-1}^T(n-1)\mathbf{f}_{m-1}(n) \end{aligned}$$

Taking the derivative of the above expression with respect to K_m:

$$\frac{\partial J_m}{\partial K_m} = 2K_m(||\mathbf{f}_{m-1}(n)||^2 + ||\mathbf{b}_{m-1}(n)||^2) + 4\mathbf{b}_{m-1}^T(n-1)\mathbf{f}_{m-1}(n)$$

Therefore, the optimal K_m minimizing the power of forward and backward errors can be chosen by equaling the derivative to zero:

$$K_m = -\frac{2\mathbf{b}_{m-1}^T(n-1)\mathbf{f}_{m-1}(n)}{||\mathbf{f}_{m-1}(n)||^2 + ||\mathbf{b}_{m-1}(n)||^2}$$

What this means basically is that the reflection coefficients (controlling PEF and prediction error itself) are computed based on all the previous samples. The idea to make this process adaptive is to introduce "forgetting" parameter $0 < \beta < 1$ so that the denominator estimate at a current sample n becomes

$$E_{m-1}(n) = ||\mathbf{f}_{m-1}(n)||^2 + ||\mathbf{b}_{m-1}(n)||^2 = \beta E_{m-1}(n-1) + (1-\beta)(f_{m-1}^2(n) + b_{m-1}^2(n-1))$$

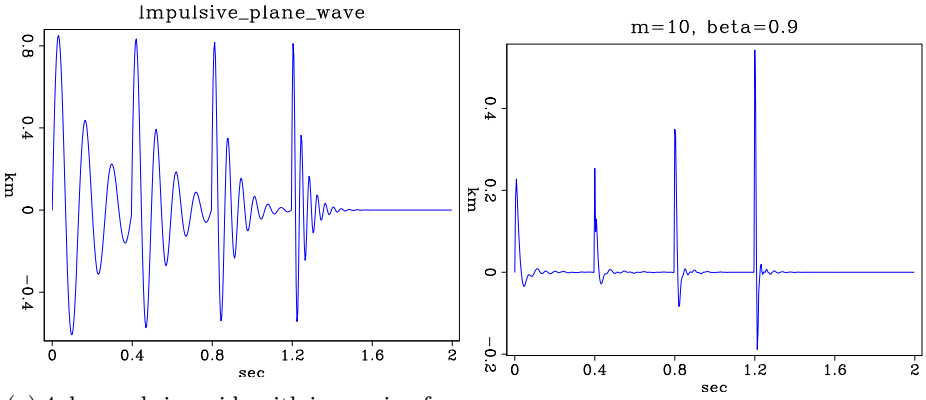
Treating the numerator in the same manner and allowing reflection coefficients to depend on the sample number $K_m(n)$ it is possible to obtain the recursive formula (HAYKIN)

$$K_m(n) = K_m(n-1) - \frac{1-\beta}{E_{m-1}(n)}(f_{m-1}(n)b_m(n) + b_{m-1}(n-1)f_m(n)) \tag{3}$$

which is similar to the update equations of normalized LMS algorithm (GRADIENT ADAPTIVE LATTICE ALGORITHM SUITABLE FOR FIXED POINT IMPLEMENTATION). This

allows algorithm to respond accordingly depending on the powers of forward and backward errors – if the errors are small ($E(n)$ is small) then the step-size $(1-\beta)/E(n)$ is large and the filter adapts rapidly, if the errors are large, the step-size is small and doesn't "respond" to variations as fast.

First of all to imitate nonstationarity, I created a simple single trace with 4 damped sinusoids with 4 distinct frequencies increasing with time (Figure 1). We can see that after filter was applied, the signal indeed has been compressed, the amplitudes, however, are not really reliable.

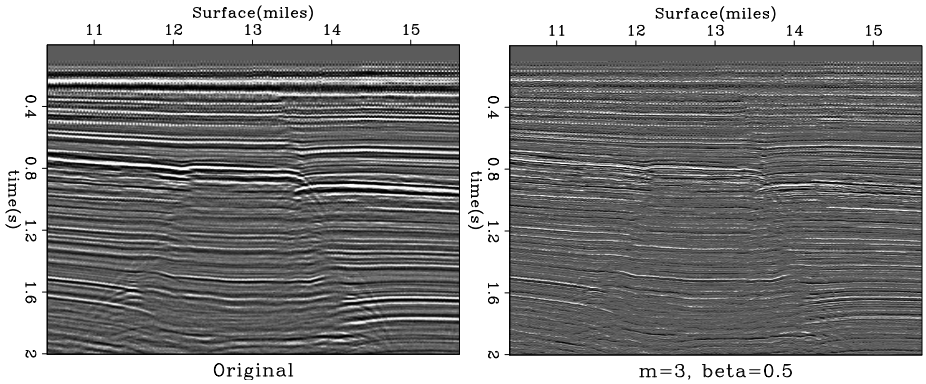


(a) 4 damped sinusoids with increasing frequencies.

(b) Result of lattice filtering.

Figure 1: Lattice filter applied to single nonstationary trace

After that I have tried applying this filter to the real stacked data on Figure 2.



(a) Raw stack.

(b) Result of lattice filtering.

Figure 2: Lattice filter applied to raw stack.

Adaptive lattice filter seems to be doing what it was expected to do. We see, that in both simple examples the temporal resolution increases significantly.

Multidimensional lattice filter

Lattice filters are implicitly solving the problem of finding inverse of autocorrelation matrix of the signal involved in the normal equation of linear prediction. Therefore, as any PEF they are carrying information of the signal spectra. Multidimensional filters are very similar and probably even more important for the real applications.

Extending lattice filters to two dimension is not a trivial task. There are several successful attempts of solving this problem (e.g. LATTICE PARAMETER AUTOREGRESSIVE MODELING OF TWO-DIMENSIONAL FIELDS). If in one-dimensional case there were two fields involved (forward and backward prediction errors), in 2D there will be even more fields that will need to be updated (4 in case of 2x2 filter, see FIGURE). Naturally, the reflection coefficients will now become vectors depending on the filter size. Finding optimal values for them in this case will involve inversion of the matrix at every lattice stage m and at every sample n.

However, it was shown in Two-Dimensional Orthogonal Lattice Structures for Autoregressive Modeling of Random Fields that problem of finding multidimensional lattice may be solved using one-dimensional lattice approach.

The first step is to sort the signal into 1D array. After this, the first iteration of the algorithm is to combine all the neighboring pairs within this 1D vector to give the first estimate of the forward and backward errors at all the points. The second iteration is combining the pair of points jumping across one sample, the third - -jumping across two samples, and so on. It continues until the jump is equal to the whole filter length (FIGURE). To make this filter adaptive, the reflection coefficients are updated by the same equation 3.

FIGURES WITH RESULTS

Recursive least-square (RLS) filter

Recursive least-square algorithm is a general extension of the least-square method that allows computation of adaptive filters as the data streams through the filter. It is well known to have better convergence properties (towards the optimal Wiener solution) compared with LMS algorithm due to the intrinsic use of inverse correlation matrices.

The first step to allow RLS filter adaptation is to introduce forgetting parameter $0 < \lambda \leq 1$ in the error-norm estimate at a given sample n:

$$E(n) = \sum_{i=1}^{n} \lambda^{n-i} |e(i)|^2 = \sum_{i=1}^{n} \lambda^{n-i} |d(i) - y(i)|^2 = \sum_{i=1}^{n} \lambda^{n-i} |d(i) - \mathbf{w}^T(n)\mathbf{u}(i)|^2$$

where $d(i)$ is the desired response at i-th sample, $y(i) = \mathbf{w}^T(n)\mathbf{u}(i)$ – is the output of adaptive filter $\mathbf{w}(n) = [w_0(n), w_1(n), \ldots, w_{m-1}(n)]^T$ estimated at sample n acting on the past m input samples $\mathbf{u}(i) = [u(i), u(i-1), \ldots, u(i-m+1)]^T$.

Due to the ill-posed nature of the problem, regularization term is included in the cost function:

$$E(n) = \sum_{i=1}^{n} \lambda^{n-i} |e(i)|^2 + \delta\lambda^n ||\mathbf{w}(n)||^2$$

Taking the derivative of this cost function with respect to filter coefficients $\mathbf{w}$ will lead to normal equations with autocorrelation matrix $\mathbf{R}(n)$, and cross-correlation $\mathbf{x}(n)$ of the desired output $d(i)$ with the input vector $\mathbf{u}(i)$:

$$\begin{aligned}
&\mathbf{R}(n)\mathbf{w}(n) = \mathbf{x}(n) \\
&\mathbf{R}(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i)\mathbf{u}^T(i) + \delta\lambda^n \mathbf{I} \\
&\mathbf{x}(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i) d(i)
\end{aligned}$$

It's easy to see now that adding the regularization term has essentially the effect of adding white noise to the data and making the correlation matrix diagonally loaded to assure its stable inverse. It is also easy to note following recursion relations (HAYKIN):

$$\begin{aligned}
&\mathbf{R}(n) = \lambda\mathbf{R}(n-1) + \mathbf{u}(n)\mathbf{u}^T(n) \\
&\mathbf{x}(n) = \lambda\mathbf{x}(n-1)\mathbf{u}(n)d(n)
\end{aligned}$$

Computing the optimal filter weights will require inverse correlation matrix $\mathbf{R}^{-1}(n) = \mathbf{P}(n)$. An efficient way of finding it is using the above recursion and Woodbury matrix inversion lemma. The resulting formulas will turn out to be (HAYKIN):

$$\begin{aligned}
&\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^T(n)\mathbf{P}(n-1) \\
&\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)}{1 + \lambda^{-1}\mathbf{u}^T(n)\mathbf{P}(n-1)\mathbf{u}(n)} \\
&\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}(n)[d(n) - \mathbf{u}^T(n)\mathbf{w}(n-1)] = \mathbf{w}(n-1) + \mathbf{k}(n)\xi(n)
\end{aligned} \tag{4}$$

Here, $\xi(n)$ is a priori estimation error – the error at a current sample n estimated using the filter weights from the previous sample $n-1$.

The equations 4 constitute the RLS algorithm. We start off from the initial filter $\mathbf{w}(0) = \mathbf{0}$ and initial correlation matrix $\mathbf{R}(0) = \delta\mathbf{I}$. Then after choosing forgetting parameter λ the algorithm proceeds giving the prediction errors.