WEIGHTED BURG-LEVINSON RECURSION WITH NOISE

Jon F. Claerbout

An earlier paper (SEP-10, p. 90, "LEVITY: Levinson Recursion Reprogrammed") showed how the Levinson shaping filter calculation could be reorganized to resemble the Burg spectral estimation procedure. Claimed advantages relate to (1) more general treatment of end effects, (2) norms other than least squares, (3) twice the numerical precision, (4) filters with positive coefficients, and (5) "tilt-invariant" solutions (for non-stationary series). What was overlooked was the occasional need to "add a little white noise" or "boost the zero lag of the cross correlation." This turned out to be important in the shot waveform estimation work that I have been doing with Raul Estevez. Basically we were working with the idea that $MS \approx -P^2$ where M is the first-order seafloor multiple, S is the unknown shot waveform, and P^2 is the seafloor primary convolved upon itself. In this problem we wanted to estimate and look at the shot waveform. It was not enough just to get the power in $MS + P^2$ to be small because noisy high and low frequencies would cause corresponding garbage in S. So the LEVITY algorithm was reorganized to incorporate parameters which, like white noise in M, tend to drive down the total power (and hence the noise power) in S while still maintaining a low power in $MS + P^2$.

The first thought was to define a process which would minimize

$$E = \min_{s} |x * s - y|^2 + \lambda |s|^2 \ , \tag{1}$$

where x and y are time series (representing, say, M and P^2), "*" denotes convolution, s is the estimated filter, and λ is an adjustable parameter that controls the importance of minimizing $|s|^2$ compared to $|x * s - y|^2$. It turned out to be easy to define a more general process that would be similar to minimizing the quadratic form

$$E = \sum_{i=1}^{N} w_i \left|(x * x - y)_i\right|^2 + \sum_{i=N+1}^{N+LC} w_i (s_i - \bar{s}_i)^2, \qquad (2)$$

where w_i are weights and $\bar{s}_i$ is the default solution. In our application, $\bar{s}_i$ was an average of earlier estimations of s_i.

In the event that the w_i are equal to one another within each of the sums in (2), and $\bar{s} = 0$, then the minimization (2) is exactly achieved. When the w_i are variable and $\bar{s} \neq 0$, it turns out that (2) probably won't be exactly minimized. In fact, I don't know exactly what the algorithm (sequential orthogonalizations) does achieve. But from inspection of the examples, I am satisfied with the results.

The algorithm works like the Burg recursion in SEP-10 (p. 90), but beyond the ends of the data x we append a delta function. Two copies of the data are made. One evolves into the forward prediction error, the other into the backward error. The delta function at the end of the first evolves into the forward prediction error filter and the one at the end of the second evolves into the backward filter. The end of the negative of the desired output y has concatenated to it the negative of the default filter $\bar{s}$. As the recursion proceeds, $-y$ evolves into $x * s - y$ and $-\bar{s}$ evolves into $s - \bar{s}$.

```
      subroutine pop(nn,n,x,ep,em,y,dp,w,lc,sbar,s)
c     inputs
      dimension x(n),y(n),w(nn),sbar(lc)
c     outputs
      dimension ep(nn),em(nn),dp(nn),s(lc)
c
c            n                                2   n+lc=nn              2
c     min   sum w(i)*(x convolve s - y)   +    sum  w(i)*(s-sbar)
c           i=1                                i=n+1
c
      nl=n+1
      do 10 i=1,n
      dp(i)=-y(i)
      ep(i)=x(i)
 10   em(i)=x(i)
      ep(nl)=1.
      em(nl)=1.
      do 20 i=1,lc
 20   dp(i+n)=-sbar(i)
      do 50 jp=1,lc
      if(jp.eq.1) go to 40
      ep(n+jp)=0.
      em(n-jp+2)=0.
      call dot(nl,ep(jp),ep(jp),w(jp),dpp)
      call dot(nl,em,em,w(jp),dmm)
      call dot(nl,ep(jp),em,w(jp),dpm)
      cp=dpm/dpp
      cm=dpm/dmm
      do 30 i=1,nl
      j=i+jp-1
      ept=ep(j)-em(i)*cm
      em(i)=em(i)-ep(j)*cp
 30   ep(j)=ept
 40   continue
      call dot(nl,dp(jp),em,w(jp),dde)
      call dot(nl,em,em,w(jp),dmm)
      cde=dde/dmm
      do 50 i=1,nl
      j=i+jp-1
 50   dp(j)=dp(j)-em(i)*cde
      do 60 i=1,lc
 60   s(i)=dp(n+i)+sbar(i)
      return
      end
      subroutine dot(n,x,y,w,ans)
      dimension x(n),y(n),w(n)
      ans=0.
      do 10 i=1,n
 10   ans=ans+w(i)*x(i)*y(i)
      return
      end
```

FIGURE 1.—Subroutine for Burg-Levinson recursion with noise.

```
c       test prog for pop: Burg-Levinson recursion with noise.
        dimension x(6),y(6),ep(9),em(9),dp(9),w(9),sbar(3),s(3)
100     type 76
        n=6
        lc=3
        nn=n+lc
        read(5,76)x
        read(5,76)y
        read(5,76)w
        read(5,76)sbar
76      format(9f3.0)
        type 81,x
        type 82,y
        type 83,w
        type 84,sbar
        call pop(nn,n,x,ep,em,y,dp,w,lc,sbar,s)
        type 87,dp
        type 88,s
        go to 100
81      format('input    x =',9f6.2)
82      format('desired  y =',9f6.2)
83      format('weights  w =',9f6.2)
84      format('default sb =',36x,3f6.2)
85      format('forward ep =',9f6.2)
86      format('backwrd em =',9f6.2)
87      format('(x*s-y)=dp =',9f6.2)
88      format('answer   s =',36x,3f6.2)
        end
```

FIGURE 2.—Test program and input test cases for Burg-Levinson recursion.

```
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1  1  1  0  0  0
 0  0  0
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1  1  1  1  1  1
 1 40-16
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1  1  1  2  3  4
 1 40-16
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1 99  1  1  1  0  0  0
 0  0  0
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1 99  1  1  0  0  0
 0  0  0
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1 99  1  0  0  0
 0  0  0
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1  1 99  0  0  0
 0  0  0
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1  1  1 99  1  1
 0  0  0
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1  1  1  1 99  1
 0  0  0
 0  0  2  1  0  0
 0  0  0 85  0  0
 1  1  1  1  1  1  1  1 99
 0  0  0
```

```
input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00  1.00  1.00  0.00  0.00  0.00
default  sb =                                      0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00  2.00 -4.00  8.00-16.00  1.00 40.00-16.00
answer    s =                                      1.00 40.00-16.00

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00
default  sb =                                      1.00 40.00-16.00
(x*s-y)=dp =  0.00  0.00  2.00 -4.00  8.00-16.00  0.00 -0.00  0.00
answer    s =                                      1.00 40.00-16.00

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00  1.00  1.00  2.00  3.00  4.00
default  sb =                                      1.00 40.00-16.00
(x*s-y)=dp =  0.00  0.00  2.00 -2.87  8.60-15.98  0.00  0.57  0.02
answer    s =                                      1.00 40.57-15.98
```

FIGURE 3.—Test program results. The first test case shows the old example of finding that the three-term filter (1, 40, -16)/85 is least-squares best for converting (2,1) to (0,1,0,0). The second example shows that if $\overline{s}$ is taken to be the s from the first example, then the result will be $s = \overline{s}$. The third example shows that non-uniform weights mess this up a bit.

```
input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00 99.00  1.00  1.00  1.00  0.00  0.00  0.00
default sb =                                       0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00  0.03-16.86  6.76-13.65  0.02 34.06-13.65
answer    s =                                      0.02 34.06-13.65

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00 99.00  1.00  1.00  0.00  0.00  0.00
default sb =                                       0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00  9.95  0.01 39.91 -0.05  4.98 40.02 -0.05
answer    s =                                      4.98 40.02 -0.05

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00 99.00  1.00  0.00  0.00  0.00
default sb =                                       0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00 31.32-65.81 -0.33 -1.05 15.66  1.76 -1.05
answer    s =                                     15.66  1.76 -1.05

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00  1.00 99.00  0.00  0.00  0.00
default sb =                                       0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00  7.85-15.71 31.41 -0.63  3.93 32.68 -0.63
answer    s =                                      3.93 32.68 -0.63

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00  1.00  1.00 99.00  1.00  1.00
default sb =                                       0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00  0.43-27.90  9.37 -9.54  0.21 28.45 -9.54
answer    s =                                      0.21 28.45 -9.54

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00  1.00  1.00  1.00 99.00  1.00
default sb =                                       0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00 27.42-68.44  1.09 -0.17 13.71  1.43 -0.17
answer    s =                                     13.71  1.43 -0.17

input     x =  0.00  0.00  2.00  1.00  0.00  0.00
desired   y =  0.00  0.00  0.00 85.00  0.00  0.00
weights   w =  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00 99.00
default sb =                                       0.00  0.00  0.00
(x*s-y)=dp =  0.00  0.00 10.50-26.24 25.73 -0.51  5.25 26.76 -0.51
answer    s =                                      5.25 26.76 -0.51
```

FIGURE 4.—Results of a weighting function which is uniform except for a strong weight somewhere in either $(x * s - y)$ or in s.