*Stanford Exploration Project, Report SERGEY, November 9, 2000, pages 259–**??***

Stanford Exploration Project, Report SERGEY, November 9, 2000, pages 259–??

Wilson-Burg spectral factorization with application to helix filtering

Paul Sava, James Rickett, Sergey Fomel, and Jon Claerbout[1]

ABSTRACT

Spectral factorization methods are used for the estimation of minimum - phase time series from a given power spectrum. We present an efficient technique for spectral factorization, based on Newton's method. We show how to apply the method to the factorization of both auto and cross-spectra, and present a simple example of 2-D deconvolution in the helical coordinate system.

INTRODUCTION

Geophysical spectral factorization is the task of estimating a minimum-phase signal with a given power spectrum. The advent of the helical coordinate system (Claerbout, 1997) has led to renewed interest in spectral factorization algorithms, since they now apply to multi-dimensional problems. Specifically, spectral factorization algorithms provide the key to rapid multi-dimensional recursive filtering with arbitrary functions, which in turn has applications in preconditioning inverse problems, and wavefield extrapolation.

The Kolmogoroff algorithm (Kolmogoroff, 1939; Claerbout, 1976) is widely used by the geophysical community since it is the most computationally efficient. However, it is not without its problems: as with all frequency domain methods, it assumes circular boundary conditions. Time-domain functions, especially those with zeros close to the unit circle, often require extreme amounts of zero-padding before they can be safely factored.

The Wilson-Burg method, introduced below, provides an algorithm for spectral factorization that is based on Newton's iteration for square-roots. Despite its iterative nature, we show that convergence is quadratic, providing a time-domain algorithm that is potentially cheaper than Kolmogoroff. In addition we describe how the algorithm can be extended to factor cross-spectra.

[1] **email:** paul@sep.stanford.edu,james@sep.stanford.edu, sergey@sep.stanford.edu, jon@sep.stanford.edu

THEORY

Newton's iteration for square roots

Let $g(a) = s - f(a)$ be the function whose roots we seek to find, where $f(a)$ is an arbitrary function and s is a constant. If we find the roots of $g(a) = 0$, then we also have found the roots of $f(a) = s$. Newton's iteration for $g(a) = 0$ can be written as $g(a_t) = -g'(a_t)(a_{t+1} - a_t)$, or

$$s - f(a_t) = f'(a_t)(a_{t+1} - a_t) \tag{1}$$

If we now consider $f(a) = a^2$, then we can write (1) as:

$$s - a_t^2 = 2a_t(a_{t+1} - a_t) \tag{2}$$

which gives Newton's iterative procedure for finding square roots

$$a_{t+1} = \frac{1}{2}\left(a_t + \frac{s}{a_t}\right). \tag{3}$$

Spectra factorization

Suppose now that we want to apply the same procedure to obtain the factors of spectral functions of $Z = e^{i\omega\Delta t}$ (Z-transforms). Let $S(Z)$ be the auto-correlation that we seek to factor into causal and anticausal parts. Burg (1998, personal communication) recognized that we can use the Newton method to factor $F(Z) = A(Z)\bar{A}(1/Z)$ by writing an equation equivalent to (2):

$$S(Z) - A_t(Z)\bar{A}_t(1/Z) = A_t(Z)[\bar{A}_{t+1}(1/Z) - \bar{A}_t(1/Z)] + \bar{A}_t(1/Z)[A_{t+1}(Z) - A_t(Z)] \tag{4}$$

If we now divide (4) by $\bar{A}_t(1/Z)A_t(Z)$ we obtain

$$\frac{\bar{A}_{t+1}(1/Z)}{\bar{A}_t(1/Z)} + \frac{A_{t+1}(Z)}{A_t(Z)} = 1 + \frac{S(Z)}{\bar{A}_t(1/Z)\; A_t(Z)} \tag{5}$$

Equation (5) leads to the Wilson-Burg algorithm:

1. Compute the right side of (5) by polynomial division forwards and backwards and then add 1.

2. Abandon negative lags, to only keep the positive powers of the Z polynomial, and also keep half of the zero lag. Now you have $A_{t+1}(Z)/A_t(Z)$.

3. Multiply out (convolve) the denominator $A_t(Z)$. Now we have the desired result $A_{t+1}(Z)$.

Iterate as long as you wish.

Quadratic convergence

The iteration converges quadratically starting from any real initial guess a_0 except zero. When a_0 is negative, Newton's iteration converges to the negative square root. Quadratic convergence means that the square of the error $a_t - \sqrt{s}$ at one iteration is proportional to the error at the next iteration

$$a_{t+1} - \sqrt{s} \quad \sim \quad (a_t - \sqrt{s})^2 = a_t^2 - 2a_t\sqrt{s} + s \quad > \quad 0 \tag{6}$$

so, for example if the error is one significant digit at one iteration, at the next iteration it is two digits, then four, etc. We cannot use equation (6) in place of the Newton iteration itself, because it uses the answer $\sqrt{s}$ to get the answer a_{t+1}, and also we need the factor of proportionality. Notice, however, if we take this factor to be $1/(2a_t)$, then $\sqrt{s}$ cancels and equation (6) becomes itself the Newton iteration (3).

Even though we cannot estimate the rate of convergence by $a_t - \sqrt{s}$ because we don't know the answer s, we can get an estimate of it by looking at the difference between the intermediate solutions at two consecutive steps. From (3), we can write

$$a_{t+1} - a_t = \frac{1}{2}\left(a_t + \frac{s}{a_t}\right) = \frac{a_t^2 + s}{2a_t} - a_t \tag{7}$$

therefore

$$a_{t+1} - a_t = \frac{s - a_t^2}{2a_t}. \tag{8}$$

Another interesting feature of the Newton iteration is that all iterations (except possibly the initial guess) overestimate the ultimate square root. This is obvious from equation (6).

Minimum phase factors

The Wilson-Burg method of spectral factorization generates minimum phase factors. Wilson (1969) presents a rigorous proof. Here is an intuitive explanation: Both sides of (5) are positive. Both terms on the right are positive. Since the Newton iteration always overestimates, the 1 dominates the rightmost term. After masking off the negative powers of Z (and half the zero power), the right side of (5) adds two wavelets. The 1/2 is wholly real, and hence its real part always dominates the real part of the rightmost term. Thus (after masking negative powers) the wavelet on the right side of (5) has a positive real part, so the phase cannot loop about the origin. This wavelet multiplies $A_t(Z)$ to give the final wavelet $A_{t+1}(Z)$ and the product of two minimum-phase wavelets is minimum phase.

Extension to cross-spectra

The same procedure as the one described above can be applied to cross-spectra. Let us first rewrite (1) as:

$$f(a_t) = s + f'(a_t)(a_t - a_{t+1}) \tag{9}$$

This is clearly nothing but a first order Taylor expansion around (s, a_{t+1}). We can write a similar relation for a two-dimensional function as:

$$f(a_t, b_t) = s + f^{(a)}(a_t, b_t)(a_t - a_{t+1}) + f^{(b)}(a_t, b_t)(b_t - b_{t+1}) \tag{10}$$

where $f^{(a)}$ and $f^{(b)}$ denote the partial derivative with respect to a and b.

If we now consider $f(a,b) = ab$, we can write (10) as:

$$a_t b_t - s = b_t(a_t - a_{t+1}) + a_t(b_t - b_{t+1}) \tag{11}$$

Now we can again use Burg's observation (1998, personal communication) and use (11) to factorize cross-spectra $F(Z) = A(Z)\bar{B}(1/Z)$ written as polynomials in $Z = e^{i\omega\Delta t}$:

$$S(Z) - A_t(Z)\bar{B}_t(1/Z) = A_t(Z)[\bar{B}_{t+1}(1/Z) - \bar{B}_t(1/Z)] + \bar{B}_t(1/Z)[A_{t+1}(Z) - A_t(Z)] \tag{12}$$

After dividing both sides by $A_t(Z)\bar{B}_t(1/Z)$, we obtain the equation that enables us to find both the causal and the anticausal part of a cross-spectrum with the Wilson-Burg algorithm:

$$\frac{\bar{B}_{t+1}(1/Z)}{\bar{B}_t(1/Z)} + \frac{A_{t+1}(Z)}{A_t(Z)} = 1 + \frac{S(Z)}{\bar{B}_t(1/Z)\, A_t(Z)} \tag{13}$$

Comparison of Wilson-Burg and Kolmogoroff methods

The Kolmogoroff algorithm (Kolmogoroff, 1939; Claerbout, 1976) is the most widely used spectral factorization algorithm because of its computationally efficiency.

Kolmogoroff method takes $O(N \log N)$ operations, where N is the length of the auto-correlation function. The cost of factorizing with the Wilson-Burg, is proportional to the [number of iterations] $\times$ [filter length] $\times N$. If we keep the filter small and limit the number of iterations, the Wilson-Burg method can be cheaper (linear in N).

Additionally, since the Kolmogoroff method works in the frequency domain, it assumes circular boundary conditions. Auto-correlation functions, therefore, need to be padded with zeros before they are Fourier transformed. For functions with zeros near the unit circle, the padding may need to be many orders of magnitude greater than the original filter length, N (Rickett and Claerbout, 1998). Wilson-Burg works in the time-domain, so no padding is needed.

Quadratic convergence means Newton's method converges extremely fast, when the initial guess is close to the solution. If we take advantage of this fact (for example, in wave extrapolation, when going from one layer to another), the method might converge in 1 or 2 iterations, reducing the cost even further. There is no way to make use of an initial guess with the Kolmogoroff method.

For 3-D filters, N may be very large, and we may need to factor many filters to account for non-stationarity for example. In these cases, the difference in cost between the two methods may be very important.

The Kolmogoroff method, when applied to helix filtering, involves the dangerous business of truncating the filter coefficients to reduce the size of the filter. If the auto-correlation function has roots close to the unit circle, truncating filter coefficients may easily lead to non-minimum-phase filters. With Wilson-Burg, we can fix the shape of the filter from the very beginning. This doesn't guarantee that we will find the exact solution, but at least we can obtain a reasonable minimum-phase approximation of the desired filter.

In many applications, we need variable (non-stationary) filtering. We believe that extending Wilson-Burg to variable filters should be trivial: just replace all the polynomial multiplications and divisions with their non-stationary equivalents.

Discussion

At this point, we would like to express a word of caution and remind the reader of some well known facts (Claerbout, 1976):

1. Not all functions are possible auto or cross-spectra.
2. Every given auto or cross-correlation function has an infinite number of possible solutions, of which there is a unique minimum phase wavelet (or pair of wavelets for cross-correlations) except for a complex scale factor of unit magnitude.
3. Let $S(Z)$ be the Z transform representation of an auto-correlation function. Suppose that we have a way to determine the roots of such a polynomial of order $2N$: $Z^N S(Z) = 0$. We can now check to see if the roots come in pairs Z and $1/Z$, i.e. if there is a root outside of the unit circle for every root inside. If this is true, we have a spectrum, and the roots outside of the unit circle are the actual roots of the minimum phase factor (Figure 1). If we don't have roots mirroring each other with respect to the unit circle, then we don't really have a spectrum, and we shouldn't even try to factorize it (Figure 2).

   A special case occurs when we have a pair of roots on the unit circle (see, for example, $Z = 1$, Figure 1). A polynomial of this type can be factorized into minimum phase causal and anticausal parts, but the convergence of the Wilson-Burg algorithm becomes linear instead of quadratic and it may even get unstable, as originally pointed out by Wilson (1969).

EXAMPLE

In a first example, we applied the method to deconvolution on the helix (Claerbout, 1997) using the factors obtained with the Wilson-Burg spectral factorization. We take the auto-correlation to be the negative of the Laplacian operator, and convolve it with a spike placed in the middle of each panel in Figure 3. We use the Wilson-Burg method to find the wavelet with this auto-correlation and then deconvolve (do polynomial division) on the helix to find back the input spike.

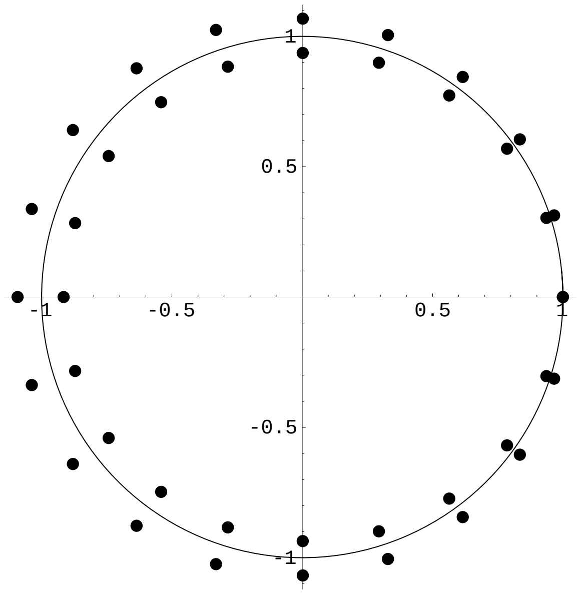

Figure 1: Roots of the Z transform representation of a Laplacian: $P(Z)=Z^N(-Z^{-N}-Z^{-1}+4-Z^1-Z^N)$. No roots are on the unit circle, except for $Z=1$ and all the roots appear in pairs that mirror each other with respect to the unit circle. burg-lapl4 [CR]

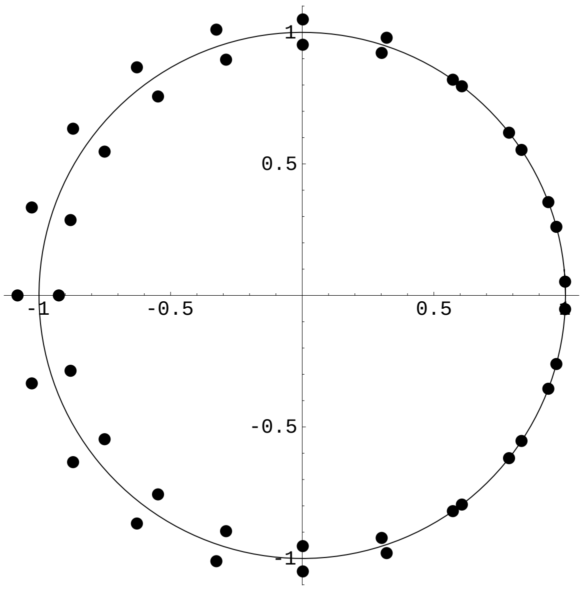

Figure 2: Roots of the Z transform representation of a pseudo-Laplacian: $P(Z) = Z^N(-Z^{-N} - Z^{-1} + 3 - Z^1 - Z^N)$ Some of the roots are on the unit circle, therefore this is not a spectrum, and so it is not factorizable. burg-lapl3 [CR]

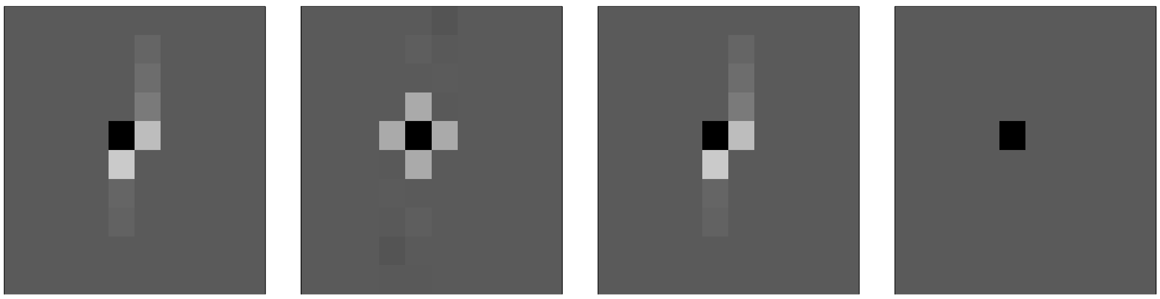

Figure 3: Wilson factorization of the Laplacian. From left to right: the input filter; its autocorrelation; the factors obtained with the Wilson-Burg method; the result of the deconvolution using the Wilson-Burg factors. burg-autolapfac [ER]

iter	A	B
1	0.0364715122	0.0442032255
2	0.0029259326	0.0002011458
3	0.0000014305	0.0000000199
4	0.0000000894	0.0000000199
5	0.0000000596	0.0000000000
6	0.0000000000	0.0000000000

Table 1: Convergence rate

In another example, we analyzed the rate of convergence of the Wilson-Burg method. We selected a simple polynomial which is the cross-correlation of two triangle functions,

$$4/Z + 15 + 11Z + 7Z^2 + 3Z^3 = 12\left(1 + \frac{1}{3Z}\right)\left(1 + \frac{3}{4}Z + \frac{1}{2}Z^2 + \frac{1}{4}Z^3\right) \tag{14}$$

Table 1 shows the quadratic rate of convergence, defined using a relation similar to equation (8) for the coefficients of the two factors, A and B.

CONCLUSIONS

We presented a method of spectral factorization that is based on Newton's iterations for square roots. The method is very efficient from a computational point of view and can be applied to both auto and cross spectra. We presented a simple example of 2-D deconvolution on a helix. Given its efficiency and simplicity, the Wilson-Burg method has the potential to surpass the Kolmogoroff method as the best method of spectral factorization. Special care must be given to the analysis of the factorizable or non-factorizable functions.

ACKNOWLEDGMENTS

We wish to thank John Burg for bringing the Wilson-Burg method to our attention and for the many useful and interesting discussions.

REFERENCES

Claerbout, J., 1976, Fundamentals of Geophysical Data Processing: http://sepwww.stanford.edu/sep/prof/.

Claerbout, J., 1997, Multidimensional recursive filters via a helix: SEP–**95**, 1–13.

Kolmogoroff, A. N., 1939, Sur l'interpolation et l'extrapolation des suites stationnairres: C.R. Acad.Sci., **208**, 2043–2045.

Rickett, J., and Claerbout, J., 1998, Helical factorization of the Helmholtz equation: SEP–**97**, 353–362.

Wilson, G., 1969, Factorization of the covariance generating function of a pure moving average process: SIAM J. Numer. Anal., **6**, no. 1, 1–7.

APPENDIX A

We use the code below to factor cross-correlation functions into two minimum phase filters. The module contains three subroutines, one for initialization, one for the actual factorization and a third one for deallocating the memory.

Type `cfilter` is a complex filter defined in the helical coordinate system. Such a filter is composed of a "zero lag" coefficient (e.g. `s0`) and two vectors, one for the lags (e.g. `ss%lag`) and one for the complex values of the filter (e.g. `ss%flt`).

We loop for the desired number of iterations. At each step, we take the auto-correlation (`ss`), divide it by A and $\bar{B}$, select the filter coefficients of the positive/negative lags, and then convolve by A/B.

The result is a pair of filters (`aa` and `bb`) of type `cfilter`.

```
module ccwil {
# Factorization of complex cross-spectra using the Wilson-Burg algorithm
  use chelicon
  use cpolydiv
  use cprint
#--------------------------------------------------------------------
  integer,                              private :: n
  complex, dimension( :), allocatable, private :: cross, ab, cc, b, c
contains
  subroutine ccwil_init( nmax) {
    integer, intent(in) :: nmax; n = nmax
    allocate( cross(2*n-1), ab(2*n-1), cc(2*n-1), b(n), c(n))
    }
  subroutine ccwil_factor( niter, s0, ss, aa, bb) {
    integer,                intent( in)     :: niter
    complex,                intent( in)     :: s0
    type( cfilter),         intent( in)     :: ss
    type( cfilter),         intent(out)     :: aa, bb
    integer                                 :: i,stat
    real                                    :: eps
    logical                                 :: T=.true.,F=.false.
    cross = 0.; cross( n) = s0; cross( n+ss%lag) = ss%flt
    ab=0.0; cc=0.0; b=0.0; b(1)=1.0; c=0.0

    do i = 1, niter {
       call cpolydiv_init( aa)
       stat = cpolydiv_clop( F, F, cross, ab)  # ab = S/A
       call cpolydiv_init( bb)
```

```
      stat = cpolydiv_clop( T, F, cc   , ab)  # cc = S/(AB*')

      b( 2:n) = cc( n+1:2*n-1)/cc(n)
      eps = maxval(abs(b(2:n)))
      call chelicon_init( aa)
      stat = chelicon_clop( F, F,     b,  c)  # c = A b
      aa%flt = c(1+aa%lag)

      b( 2:n) = cc( n-1:1:-1 )/cc(n)
      eps = max(abs(maxval(abs(b(2:n)))),abs(eps))
      call chelicon_init( bb)
      stat = chelicon_clop( F, F,     b,  c)  # c = B* b
      bb%flt = c(1+bb%lag)
      write( 0,*) i, eps
      }
      bb%flt=conjg(bb%flt)
  }
  subroutine ccwil_close () {
    deallocate( cross, ab, cc, b, c) }
}
```