A ROOTFINDING ALGORITHM FOR LINE SPECTRAL FREQUENCIES

Joseph Rothweiler

Sanders, A Lockheed Martin Company
65 River Road
Hudson, NH 03051
joerothweiler@ieee.org

ABSTRACT

Published techniques for computing line spectral frequencies generally avoid rootfinding methods because of concerns about convergence and complexity. However, this paper shows that stable predictor polynomials have properties that make rootfinding an attractive approach.

It is well known that the problem of finding the LSF's for an N'th order predictor polynomial can be reduced to the problem of finding the roots of a pair of polynomails of order N/2 with real roots. I extend this result by showing that these polynomials have the following properties:

1. It is possible to select starting points for a Newton's rootfinding method such that the iteration will converge monotonically to the largest root.
2. The Newton iteration can be modified to speed up the process while still maintaining good convergence properties.

In this paper, I present the rootfinding procedures with proofs of their good convergence properties. Finally, I present experimental results showing that this procedure performs well on speech signals, and that it can be implemented on fixed-point DSP's. [1]

1. INTRODUCTION

The line spectral frequency (LSF) parameter set is an alternate representation of the parameters of an autoregressive (AR) model.The LSF representation is particularly attractive in medium to low rate voice coding systems because the parameters can be efficiently quantized. It is used in the U.S. Federal Standard CELP coder,[1] and many other voice coding algorithms.

A disadvantage of the LSF representation is that the parameters are defined as the roots of a polynomial. Root finding is usually undesirable in a real-time system because it is an iterative procedure which is subject to convergence problems, is sensitive to roundoff errors, and has unpredictable processing delays. To avoid these problems, existing systems typically use exhaustive searches and similar techniques with high — but bounded — computational complexity. A notable exception is a recent paper by Wu and Chen[2].

In this paper, I show that convergence problems can be completely avoided by taking advantage of the special structure of the polynomials that must be solved. Root-finding methods therefore become a very attractive method of finding the LSF's.

This paper is organized as follows:

Section 2 is a brief review of LSF computation, including polynomial transformations based on original work by Soong and Juang[3], which was later extended by Kabal and Ramachandran[4]

Section 3 demonstrates that the reduced polynomials have properties which guarantee that Newton's root finding method will converge monotonically to the roots. I also show that an accelerated version of Newton's method will converge faster while remaining well-behaved. Finally, I discuss deflation procedures to ensure stability of the entire root finding process.

Section 5 presents experimental results which show that the proposed method works well on AR models derived from real speech data. It also describes some results of experiments with a version of this algorithm that uses fixed-point arithmetic.

2. BASIC DERIVATION

Start with the prediction error filter of order P:

$$A(z) = 1 - \sum_{k=1}^{P} a_k z^{-k} \tag{1}$$

and define the symmetric and antisymmetric filters

$$F_s(z) \quad = \quad A(z) + z^{-P-1} A(z^{-1}) \tag{2}$$

$$F_a(z) \quad = \quad A(z) - z^{-P-1} a(z^{-1}) \tag{3}$$

Assuming that $1/A(z)$ is a stable filter, Soong and Juang showed that the roots of F_s and F_a (A) lie on the unit circle, (B) are interleaved, (C) are distinct, (D) exactly two of the roots are at $+1$ and -1.

Removing the roots at ± 1 gives

$$G_1(z) = \frac{F_s(z)}{1+z^{-1}} \quad G_2(z) = \frac{F_a(z)}{1-z^{-1}} \quad P\ even \tag{4}$$

$$G_1(z) = F_s(z) \quad G_2(z) = \frac{F_a(z)}{1-z^{-2}} \quad P\ odd \tag{5}$$

Note that in either case, G_1 and G_2 are symmetric polynomials of even order ($2M_1$ and $2M_2$), so either polymonial can be written as (after substituting $z \equiv e^{j\omega}$):

$$G(z) \quad = \quad \sum_{i=0}^{2 \cdot M} g_i e^{-ji\omega} \tag{6}$$

[1] Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002

```
void cos2x(double g, int g_order) {
   int i,j;
   for(i=2; i<= g_order; i++) {
      for(j=0; j <= g_order-i; j++) {
         g[j+2] -= g[j];
         g[j] *= 2.0;
      }
   }
}
```

Figure 1: Conversion from $Y(\omega)$ to a polynomial in x.

$$\begin{aligned} &= \sum_{i=0}^{M-1} \left[g_i e^{-ji\omega} + g_{M-i} e^{-j(M-i)\omega} \right] + g_M e^{-jM\omega} \\ &= e^{-jM\omega} \left[\sum_{i=0}^{M-1} 2 g_i cos[(M-i)\omega] + g_M \right] \\ &= e^{-jM\omega} \left[\sum_{k=0}^{M} c_k \cos(k\omega) \right] \equiv e^{-jM\omega} Y(\omega) \end{aligned} \quad (7)$$

where

$$c_0 \equiv g_M \quad and \quad c_k \equiv 2 g_{M-k}\,, \quad k = 1, \ldots M \quad (8)$$

Because the roots all have the same modulus, it is convenient to transform the polynomials from the z domain into the $x \equiv \cos(\omega)$ domain.

Soong and Juang convert $Y(\omega)$ to a polynomial in x using multiple-angle relatons, while Kabal and Ramachandran used Chebyshev polynomials. The author[5] recently proposed the following extension of this work that is based on the recursive application of Chebyshev polynomial relations.

Write equation 7 above as

$$Y(x) = c_0 T_0(x) \; + \; \ldots \; + \; c_N T_N(x) \quad (9)$$

where $x = \cos(\omega)$ and $T_k(x) \equiv \cos(k\omega) \equiv cos\left(k \cdot \cos^{-1}(x)\right)$ is the Chebyshev polynomial of order k. Using the relation

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \quad (10)$$

we can transform $Y(x)$ to the form

$$Y(x) = xY'(x) + c_0' \quad (11)$$

Where Y' is a sum of Chebyshev polynomials up to order $N-1$. Recursively applying this transformation to Y', (a total of N-1 times), we finally obtain

$$Y(x) = \sum_{k=0}^{N} c_k' x^k \quad (12)$$

The advantage of this formulation is that it suggests an exceptionally simple mechanization of the transformation, as shown in subroutine `cos2x` of Fig. 1. In this program, Eq. 10 is implemented as a single arithmetic operation. The inner loop performs the transformation of Eq. 11, while the outer loop repeats this transformation to obtain the form of Eq. 12.

After this transformation, we have a pair of polynomials in x, whose roots

1. are distinct,
2. are interleaved,
3. are real and lie in the range $(-1, +1)$.

3. NEWTON'S METHOD AND AN ACCELERATED VARIATION

Stoer and Bulirsch[6] derive several useful properties of rootfinding methods based on the following general procedure:

Given a polynomial $p(x)$ with derivative $p'(x)$, start with an initial guess x_0 for the root location, and refine the guess using the recursion

$$x_i = x_{i-1} - \beta \frac{p(x_{i-1})}{p'(x_{i-1})} \quad (13)$$

The case $\beta = 1$ is the standard Newton's iteration, while the case $\beta = 2$ corresponds to an accelerated procedure to be discussed below.

For the case β=1, Stoer and Bulirsch prove the following (Theorem 5.5.5, p. 272):

Theorem 1 *Let p(x) be a polynomial of degree $n \geq 2$ with real coefficients. If all roots ξ_i, $\xi_1 \geq \xi_2 \geq \ldots \geq \xi_n$, of $p(x)$ are real, then Newton's method yields a convergent strictly decreasing sequence x_k for any initial value $x_0 > \xi_1$.*

It has already been proven that the roots of $Y(x)$ satisfy the stated conditions on the roots, and that all roots lie in the range $(-1, 1)$. Therefore, if we start with an initial guess of 1, Newton's method will converge monotonically to the most positive root of $Y(x)$. It is easy to see that this same theorem guarantees that an initial guess of -1 will cause Newton's method to converge monotonically to the most negative root. After finding these outermost roots, we can divide them out to reduce the polynomial order by 2. This procedure can then be repeated to locate the other roots.

Stoer and Bulirsch also prove a theorem that applies to the case $\beta = 2$ (Theorem 5.5.9, p. 274):

Theorem 2 *Let $P(x)$ be a real polynomial of degree $n \geq 2$, all roots of which are real, $\xi_1 \geq \xi_2 \geq \ldots \geq \xi_n$. Let α_1 be the largest root of $p'(x)$: $\xi_1 \geq \alpha_1 \geq \xi_2$. (For $n = 2$, we also require that $\xi_1 > \xi_2$). Then for every $z > \xi_1$, the numbers*

$$z' := z - \frac{p(z)}{p'(z)}, y := z - 2\frac{p(z)}{p'(z)}, y' := y - \frac{p(y)}{p'(y)} \quad (14)$$

are well defined and satisfy

$$\alpha_1 < y\,, \quad \xi_1 \leq y' \leq z'. \quad (15)$$

This theorem is depicted graphically in Fig. 2. Here, z is the initial guess for the root, and y is the result of one double-step iteraton. There are 3 possibilities: $y = \xi_1$, $y > \xi_1$, and $y < \xi_1$. In the first case, we have found the root. In the second case (no overshoot), we can use y as the new starting point for a double-step iteration. In the final case (y overshoots the root location), we can compute y', which is a valid starting point for a single-step iteration. We know that $y > \alpha_1 \geq \xi_2$, which guarantees that overshoots can be detected by a difference in signs betwen $p(y)$ and $p(z)$.

Therefore, the algorithm for finding the largest root of g(x) is:

1. Start with an initial guess of $x_0 = 1$ for the root location.

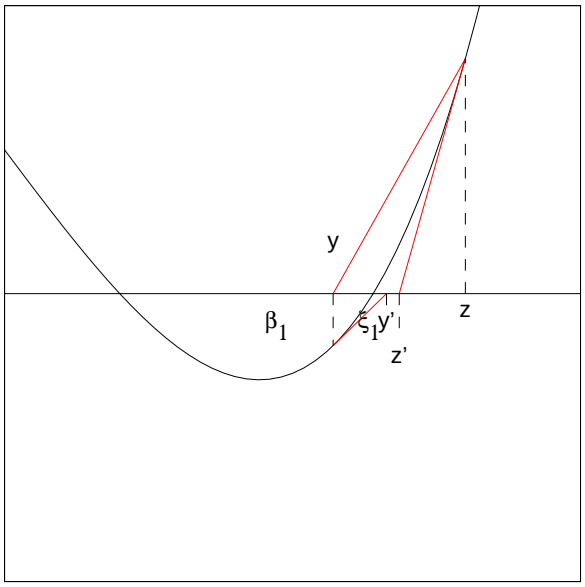

Figure 2: Graphical depiction of the convergence of the second-order Newton iteration

2. Compute an updated estimate x_{i+1} using the double-step iteration.
3. if $sign(x_{i+1}) = sign(x_i)$, increment i and repeat step 2.
4. Otherwise, begin a single-step Newton iteration starting at x_{i+1}.
5. Continue the single-step iteration until the root is located with sufficient precision.

Stopping criteria will be considered in the next section.

It is obvious that the same procedure will also converge to the most negative root of g(x) if a starting point of -1.0 is used.

3.1. Deflation.

A common problem in rootfinding is that numerical errors introduced while dividing off the initial roots can cause significant errors in the values of later roots. There are rootfinding procedures that perform all computations on the original polynomial, thus mitigating the effects of finite arithmetic precision. However, these procedures involve substantially more computation, so deflation is preferred in time- critical applications if stability can be maintained.

It can be shown that the harmful effects of deflation are minimized if (Stoer and Bulirsch, p. 278):

- The root of largest absolute value is divided off, and
- The coefficients of the deflated polynomial are computed in reverse order.

A program was written based on these constraints, and the experimental results (next section) show that it gives good results for predictor polynomials computed from speech signals.

4. THE PROGRAM.

A complete program implementing this algorithm is available on the web at `http://www.xtdl.com/řothwlr`. Tests were run to select a good stopping criterion for the root-finding iteration and to evaluate the complexity for implementation on a fixed-point digital signal processor.

4.1. Stopping criterion.

The iteration terminates when either $p^2(x_i) < T_1$ or when $(x_i - x_{i-1})^2 < T_2$. To select the best stopping criterion and the optimum threshold, a series of tests were run to evaluate the dependence of the error on the thresholds. The DR1 accent group of training data in the TIMIT data base[7] was used for testing.

To measure the accuracy of the rootfinder, the computed pole locations were substituted back into the original G_1 and G_2 polynomials and the polynomial values recomputed. The quality parameter is thus defined as

$$\eta = \sqrt{\frac{1}{N_1}\sum_{i=0}^{N_1} G_1^2(x_{1,i}) + \frac{1}{N_2}\sum_{i=0}^{N_2} G_2^2(x_{2,i})} \qquad (16)$$

Plots of error as a function of the threshold value for the first stopping criterion are shown in Figures 3. It will be seen that T_1 values between 10^{-7} and 10^{-15} give essentially the same accuracy (limited by the machine precision). Similar results were obtained for T_2.

As shown in the figure, computations were performed using both forward and reverse deflation. Results show that reverse deflation gives a slight but consistent improvement in accuracy.

4.2. Complexity.

A second set of runs was made on the same data set in order to determine the relationship between model order and complexity. The threshold was set to 10^{-8}, based on the results of the previous tests. These tests all used reverse deflation, but the rootfinder was optionally patched to use the conventional Newton iteration instead of the accelerated version.

The program as presented includes statements to count multiply-accumulate, addition- subtraction, and division operations. I am assuming that this program will typically be run on a Digital Signal Processor which has a single cycle multiply-accumulate instruction.

As plotted in Fig. 4, the complexity shows a linear relationship with model order, and worst-case complexity is only about twice the average complexity. This shows that the algorithm can be applied to real speech data without fear of excessive run times.

5. REAL-TIME IMPLEMENTATION.

A version of this program was written to simulate the effects of 16-bit arithmetic. The simulation was based on the structure of the Texas Instruments TMS320C50 Processor: all variables are stored as 16-bit numbers, and a 32-bit accumulator is used for the computations. Satisfactory operation is obtained provided that:

- All parameters are scaled so that maximum precision is retained at each step.

- Rounding is performed when accumulator values are stored into 16-bit memory.
- Reverse deflation is used.

Even with these precautions, it was found that numerical errors occasionally caused instability. These instabilities were detected by checking for non-monotonic convergence of the Newton iteration, and by the use of a loop counter to limit the maximum number of iterations.

A test program was written to compare the fixed-point and floating-point versions for a 12'th order model. On a test set of over 20 000 stable coefficient sets, only 0.3% became unstable during fixed-point processing. Listening tests confirmed that these failures had no effect on speech intelligibility. More details of the tests are available in an unpublished manuscript[8].

6. SUMMARY.

This technique provides a simple and flexible procedure for computing Line Spectral Frequencies for any model order. The program is computationally efficient and robust, and has been shown to be practical for real-time implementation on current DSP's.

7. REFERENCES

[1] Jr. Joseph Campbell, Vanoy Welch, and Thomas Tremain, "An expandable error-protected 4800 bps celp coder (u.s. federal standard 4800 bps voice coder)," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 735–738, May 1989.

[2] Chung-Hsien Wu and Jau-Hung Chen, "A novel tow-level method for the computation of the lsp frequencies using a decimation-in-degree algorithm," *IEEE Trans. on Speech and Audio Processing*, vol. 5, no. 2, pp. 106–115, March 1997.

[3] F. K. Soong and B. H. Juang, "Line spectrum pair (lsp) and speech data compression," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 1.10.1–1.10.4, May 1984.

[4] P. Kabal and R. P. Ramachandran, "Computation of line spectral frequencies using chebyshev polynomials," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 34, no. 6, pp. 1419–1426, Dec 1986.

[5] Joseph Rothweiler, "On polynomial reduction in the computation of lsp frequencies," *To be published in IEEE Trans. on Speech and Audio Processing*, 1999.

[6] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, 1983.

[7] "Timit acoustic-phonetic continuous speech corpus," Available from NTIS. Order No. PB91-505065, Oct 1990.

[8] Joseph Rothweiler, "Rootfinding methods for computing line spectral frequencies," *Unpublished Manuscript*, 1995.

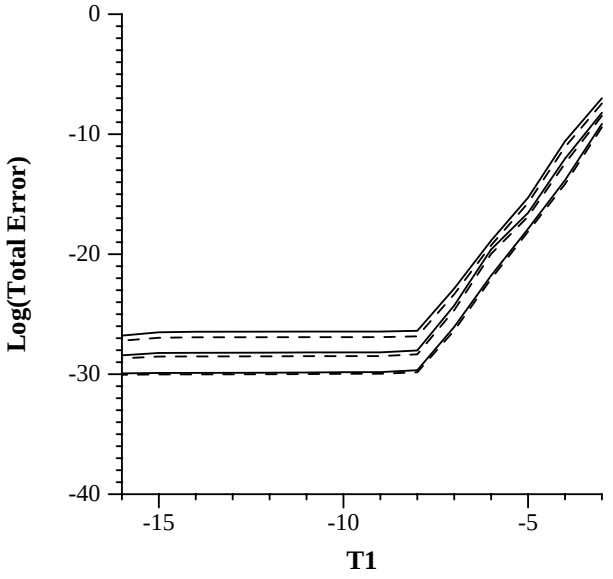

Figure 3: Log LSF error as a function of stopping threshold T_1. Solid curves are forward deflation, dashed curves are reverse deflation. Predictor orders are (top to bottom): 16, 12, 8.

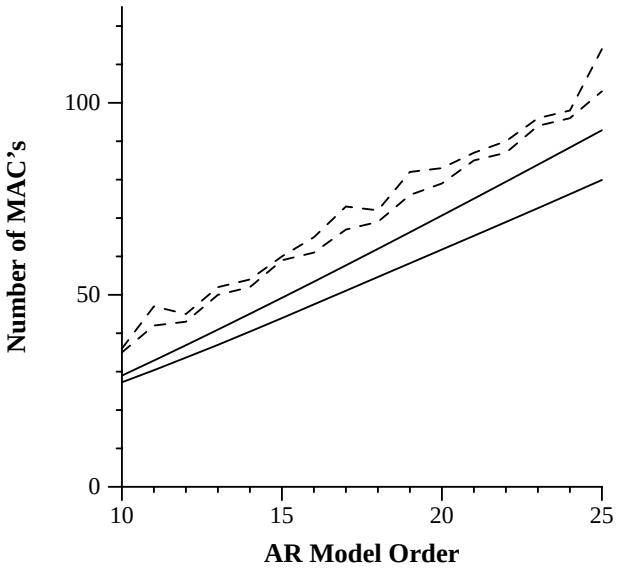

Figure 4: Complexity vs. model order. Solid curves are averages, dashed curves are maximums. For each pair, bottom curve is $\beta = 2$, top curve is $\beta = 1$