On Polynomial Reduction in the Computation of LSP Frequencies

Joseph Rothweiler
Sanders, A Lockheed-Martin Company
P.O. Box 868, PTP2-A001
Nashua, NH 03061-0868

Abstract— **This letter describes a new version of the "Decimation-in-Degree" (DID) transformation used by Wu and Chen[?] as part of their procedure for computing Line Spectrum Pair (LSP) coefficients. This new version eliminates all nontrivial multiplications, requires fewer total operations (additions and multiplications), and is performed in-place — eliminating a memory buffer.**

I. INTRODUCTION

RECENTLY, Wu and Chen[?] described a two-step algorithm for computing the line spectral frequencies. The algorithm is an extension of a method originally proposed by Soong and Juang [?] and further developed by Kabal and Ramachandran[?]. An overview of the algorithm is as follows:

1. Beginning with the minimum-phase LPC polynomial
$$A_m(z) = 1 + a_1 z^{-1} + ... + a_m z^{-m} \tag{1}$$
2. Form the symmetric and antisymmetric polynomials
$$\begin{aligned} P(z) &= A_m(z) + z^{-(m+1)} A_m(z^{-1}) \\ Q(z) &= A_m(z) - z^{-(m+1)} A_m(z^{-1}) \end{aligned} \tag{2}$$
3. Divide by the trivial zeros at $z = \pm 1$, giving even-order symmetric polynomials
$$\begin{aligned} R_P(z) = \tfrac{P(z)}{1+z^{-1}}, \quad R_Q(z) = \tfrac{Q(z)}{1-z^{-1}}, \quad m: even \\ R_P(z) = P(z), \quad R_Q(z) = \tfrac{Q(z)}{1-z^{-2}}, \quad m: odd \end{aligned} \tag{3}$$
The orders of R_P and R_Q are designated as $2p$ and $2q$,respectively.
4. Using the transformation[1] $x = z + z^{-1} = 2\cos\omega$, convert $R_P(z)$ and $R_Q(z)$ into polynomials in x:
$$\begin{aligned} F_P(x) &= \textstyle\sum_{n=0}^{p} c_n x^n \\ F_Q(x) &= \textstyle\sum_{n=0}^{q} d_n x^n \end{aligned} \tag{4}$$
All roots of R_P and R_Q are distinct and lie on the unit circle, so all roots of F_P and F_Q are distinct, real and lie in the range $[-1, +1]$.
5. Find the roots of $F_P(x)$ and $F_Q(x)$.
6. Compute the line spectral frequencies as
$$\omega_i = \arccos \frac{x_i}{2}, \quad i = 1, ..., p+q \tag{5}$$
where the x_i are the roots of F_P and F_Q.

The Wu-Chen procedure has 2 key differences from previous algorithms:

- The Decimation-In-Degree algorithm described in the paper is an efficient method for implementing step 4 that is easily adaptable to varying LPC model orders. In [?], a similar transformation is derived for an 8'th order model using cosine multiple-angle relations. Transformations for other model orders are easily derived using the same approach, but a program supporting different orders must contain a transformation for each possible order. In [?], Chebyshev polynomials are used to develop a similar transformation.
- Roots are found using a rootfinding algorithm, which is more efficient than the iterative search techniques used by most other researchers.

This letter describes a method for performing step 4 by repeated application of a single Chebyshev polynomial relation. Like DID, it is easily adaptable to varying model orders. Advantages of the new procedure over DID are:

- it can be performed in-place, eliminating the need for a separate output array and an intermediate storage array,
- it requires no multiplications (except trivial multiplications by 2 that can be implemented as an add or shift operation),
- the number of operations is lower than the number of operations in DID.

[1] Note that the other referenced papers use the definition $x = (z + z^{-1})/2 = \cos\omega$, which introduces minor scaling differences. The derivation in section 2 of this paper also uses the definition $x = 2\cos\omega$.

II. Chebyshev Polynomial Reduction

Note first that R_P and R_Q are always symmetric polynomials of even order. Therefore, we can write R_P as (the reduction for R_Q is the same, and is not shown):

$$R_P(z) = r_0 z^{-p} + \sum_{n=1}^{p} r_n (z^{-p-n} + z^{-p+n}) \qquad (6)$$

Factoring out z^{-p}, and noting that $z^n + z^{-n} = e^{jn\omega} + e^{-jn\omega} = 2\cos(n\omega)$, we have

$$R_P(\omega) = e^{-jp\omega}\left[r_0 + 2\sum_{n=1}^{p} r_n \cos n\omega\right] \equiv e^{-jp\omega} Y(\omega) \qquad (7)$$

The line spectral frequencies are defined as the frequencies ω_i for which $R_P(e^{j\omega_i}) = 0$. The ω_i are also the roots of Y. To find the roots, it is more convenient to convert Y into a polynomial instead of a sum of cosines.

To begin this conversion, let $x = 2\cos\omega$, so that

$$Y(x) = r_0 + \sum_{n=1}^{p} 2 r_n T_n(x/2) \qquad (8)$$

where $T_n(\cdot) \equiv \cos n\omega \equiv cos\left(n\cos^{-1}(\cdot)\right)$ is the Chebyshev polynomial of order n.

The Chebyshev polynomials obey the well-known recursion

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_n(x) &= 2xT_{n-1}(x) - T_{n-2}(x) \end{aligned} \qquad (9)$$

so each T_n can be expressed as a polynomial in x of order n. One way to reduce Eq. 8 is to substitute the appropriate polynomial for each T_n, and sum the coefficients of like powers of x. However, as will be shown next, term-by-term backward substitution of Eq. 9 into Eq. 8 leads to a computationally simpler way to perform this reduction.

The goal of these backward substitutions is to convert Y to the form

$$Y(x) = r'_0 + x\left[r'_1 + \sum_{n=2}^{p} 2r'_2 T_{n-1}(x/2)\right] = r'_0 + xY'(x) \qquad (10)$$

where Y' is a sum of Chebyshev polynomials up to order $p-1$. Because Y' has the same form as Y, the same transformation can be applied recursively to $Y'(x)$, reducing the Chebyshev polynomial order by 1 with each recursion, until $Y(x)$ has been converted into a polynomial in x.

Note that direct substitution of Eq. 9 into Eq. 8 results in some terms containing x and other terms not containing x, and therefore does not simplify the computation. Instead, the substitution should be performed in a backward sequence on alternate terms:

- Substitute Eq. 9 into the last term of Eq. 8, so that the p and $p-2$ terms become

$$\begin{aligned} &2r_pT_p(x/2) + 2r_{p-2}T_{p-2}(x/2) = \\ &2r_p x T_{p-1}(x/2) + 2(r_{p-2} - r_p)T_{p-2}(x/2) \end{aligned} \qquad (11)$$

  so $r'_{p-2} = r_{p-2} - r_p$.
- Next, substitute for T_{p-2}, so that the $p-2$ and $p-4$ terms become

$$\begin{aligned} &2r'_{p-2}T_{p-2}(x/2) + 2r_{p-4}T_{p-4}(x/2) = \\ &2r'_{p-2} x T_{p-3}(x/2) + 2(r_{p-4} - r'_{p-2})T_{p-4}(x/2) \end{aligned} \qquad (12)$$

  so $r'_{p-4} = r_{p-4} - r'_{p-2}$.
- Repeat the substitution for lower terms, down to T_2 or T_3.
- Perform a similar sequence of backward substitutions starting with T_{p-1}.

The T_3 and T_2 terms are special cases. For T_3,

$$\begin{aligned} &2r'_3T_3(x/2) + 2r_1T_1(x/2) = \\ &2r'_3 x T_2(x/2) + 2(r_1 - r'_3)T_1(x/2) = \\ &2r'_3 x T_2(x/2) + (r_1 - r'_3)x \end{aligned} \qquad (13)$$

so $r'_1 = r_1 - r'_3$ is computed using the same recursion as the higher terms, even though it is not multiplied by 2.

For T_2,

$$2r'_2T_2(x/2) + r_0 = 2r'_2 x T_1(x/2) + (r_0 - 2r_2) \qquad (14)$$

so

$$r'_0 = r_0 - 2r'_2 \qquad (15)$$

To summarize, the updated coefficients r'_n of Eq. 10 are computed using the recursion

$$r'_n = r_n - r'_{n+2} \quad , \qquad p-2 \geq n \geq 1 \qquad (16)$$

with the initial conditions

$$\begin{aligned} r'_p &= r_p \\ r'_{p-1} &= r_{p-1} \end{aligned} \qquad (17)$$

and final step

$$r'_0 = r_0 - 2r'_2 \qquad (18)$$

Each update in Eq. 16 can be implemented by a single addition, and the operation can be done in place because the coefficient of T_n is not needed in later steps of the recursion.

TABLE I
Operation counts for DID and Chebyshev reduction of both polynomials

Order	Wu and Chen			Chebyshev		
	mpy	add	ptr	mpy	add	ptr
6	4	8	4	4	6	0
8	8	16	8	6	12	0
10	12	24	12	8	20	0
12	18	36	18	10	30	0
14	24	48	24	12	42	0
20	50	100	50	18	90	0

The advantage of this formulation is that it suggests an exceptionally simple mechanization of the transformation, as shown in subroutine `cheby` of Fig. 2. In this program, Eq. 9 is implemented as a single arithmetic operation. The inner loop performs the transformation of Eq. 16, while the outer loop repeats this transformation to obtain the form of Eq. 10. Note that the resulting roots are in the range -2 to 2, as in [1].

III. Algorithm Comparisons

Fig. 1 is a direct implementation of the DID algorithm from [?]. The simplicity of the new implementation in Fig. 2 is evident. Note that the conversion is done in-place, eliminating the need for a second buffer to hold the results, as well as eliminating the auxiliary `s` array.

In both figures, variables `mpy` and `add` have been added to count math operations in the main part of the algorithm as done by Wu and Chen. Variable `ptr` counts array indexing operations that they included as additions, but which could be handled separately by indexing modes on many digital signal processors. Table I presents the number of operations required to convert both the R_P and R_Q polynomials (*i.e.,* two calls to each subroutine).

Complexity comparisons make implicit assumptions about the type of hardware used to implement the algorithm. In a general purpose computer being programmed in a high-level language, multiplies, adds and indexing operations are of approximately equal complexity. Most digital signal processors include a single-operation multiply-add instruction, as well as indexing and looping registers that can perform many operations in parallel with data operations. Using Field-Programmable Gate Arrays or other direct hardware implementation, multiplications tend to be more complex than additions, which in turn tend to be more complex than bit-shift operations. Therefore, relative advantages of the two algorithms may vary depending on how they are to be implemented.

Another complexity factor not considered here is fixed-point vs. floating-point implementation. The algorithm presented here has been implemented in a speech coding algorithm in fixed-point, with no measurable change in speech quality. However, additional operations are required to scale the coefficients to prevent overflow. The Wu and Chen algorithm would require similar overflow protection as well.

IV. Summary

The new algorithm provides an efficient method of performing the polynomial transformation that is a key part of LSF computation procedures such as that proposed by Wu and Chen in[?]. It can be implemented in-place, has a simple software implementation, and requires no nontrivial multiplications. It is particularly suitable for general purpose subroutine libraries and other computer programs to be used with variable LPC model orders. When the model order is fixed (such as in many speech vocoders), this algorithm still provides a simple implementation without nontrivial multiplications.

A demonstration program implementing the full transformation based on this algorithm is available on the author's web site:
`http://www.xtdl.com/~rothwlr`

V. Acknowledgements

The author would like to thank the reviewers for their comments, suggestions, and clarifications, particularly issues related to comparisons of algorithmic complexity.

```
void kw(double *r,int n) {
    double s[100];
    double c[100];
    int i, j, k;

    s[0] = 1.0;
    s[1] = -2.0;
    s[2] = 2.0;
    for(i=3;i<=n/2;i++) s[i] = s[i-2];

    for(k=0;k<=n;k++) {
        c[k] = r[k];
        j = 1;
        for(i=k+2;i<=n;i+=2) {
            c[k] += s[j]*r[i];      mpy++;add++;
            s[j] -= s[j-1];         add++;
            j++;                    ptr++;
        }
    }
    for(k=0;k<=n;k++) r[k] = c[k];
}
```

Fig. 1. ANSI C program implementing polynomial reduction as proposed by Wu and Chen[1]

```
void cheby(double *g, int ord) {
    int i, j;
    int k;

    for(i=2; i<= ord; i++) {
        for(j=ord; j > i; j--) {
            g[j-2] -= g[j];        add++;
        }
        g[j-2] -= 2.0*g[j];        mpy++; add++;
    }
}
```

Fig. 2. Improved polynomial reduction based on Chebyshev polynomials