PREDICTIVE LSF COMPUTATION

Bogdan Dumitrescu[*], *Ioan Tăbuş*
Signal Processing Laboratory,
Tampere University of Technology
P.O. Box 553, 33101 Tampere, Finland
e-mail: bogdand,tabus@cs.tut.fi

ABSTRACT

This paper presents a new method for computing the line spectral frequencies (LSF). The new features of the method consist in: (1) the use of predictions of the LSFs as starting points for the search of the current LSFs, (2) the fast bracketing of the LSFs working on a grid of points and (3) the refinement of LSFs by a variable number of bisections. The method provides an easy control of the trade off between necessary accuracy and worst case computational time and is easily tunable to different orders of prediction polynomials. The experimental results show an improvement of the average performance with more than 30% with respect to a frequently used method (Kabal-Ramachandran).

1 INTRODUCTION

Current speech coders in the range 4-32 Kbits/s model the vocal short-time spectral envelope (for each frame of sampled signal) using a linear predictor, i.e. the n-th order AR filter $1/A(z)$, with

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

The parameters of the filter are usually coded using the line spectral frequencies (LSF) [1]. Supposing that n is even, the polynomials $\tilde{P}$ and $\tilde{Q}$, defined as follows,

$$\begin{aligned} \tilde{P}(z) &= A(z) + z^{-(n+1)}A(z^{-1}) = P(z)(1+z^{-1}), \\ \tilde{Q}(z) &= A(z) - z^{-(n+1)}A(z^{-1}) = Q(z)(1-z^{-1}), \end{aligned} \tag{2}$$

are symmetric and antisymmetric, respectively. All their roots lie on the unit circle and are distinct, with trivial roots at -1 and 1, respectively. After removing the trivial factors, the symmetric polynomials $P(z)$ and $Q(z)$ are obtained, with zeros of the form $z_i = \mathrm{e}^{j\omega_i}$. Since the roots appear in complex conjugate pairs, one may restrain to the roots z_i located on the upper unit semicircle, defined by $x_i = \mathrm{Re}\ z_i$, $i = 1:n$, (named line spectral pairs – LSP) or by $\omega_i \in (0, \pi)$, $i = 1:n$, named LSF.

If the LSPs and LSFs are ordered,

$$\begin{aligned} &1 > x_1 > x_2 > \ldots > x_n > -1 \\ &0 < \omega_1 < \omega_2 < \ldots < \omega_n < \pi, \end{aligned} \tag{3}$$

then an interlacing property holds, i.e. the odd LSPs belong to P and the even LSPs to Q. The relation between LSPs and LSFs is obvious: $x_i = \cos\omega_i$.

[*]On leave from the Department of Control and Computers, Politehnica University of Bucharest, 313, Spl. Independenţei, 77206 Bucharest, Romania, e-mail: bogdan@lucky.schur.pub.ro.

The most efficient methods for computing the LSFs have two stages. In the first, two polynomials P_r and Q_r of degree $m = n/2$ and with real roots are computed from P and Q. The algorithm of Kabal and Ramachandran [3] uses a Chebyshev recurrence relation to evaluate P_r and Q_r, which are actually not formed explicitly. The algorithm of Wu and Chen [5] computes explicitly P_r and Q_r, which implies some extra operations, with the advantage of using the Horner algorithm for polynomial evaluation.

In the second stage, the roots of the reduced polynomials are computed. In [3], $P_r(x)$ is evaluated on a fine grid in order to detect sign changes; when a root is bracketed, the search is refined by a fixed number of bisections and a final linear interpolation is performed. The search starts, for example, at 1; taking advantage of the interlacing property, when a root of a polynomial is found, the search continues for a root of the other polynomial.

In [5], the (modified) Newton-Raphson method is used to find a root, then the polynomial is deflated with respect to that root. The procedure is repeated until closed-form formulas may be applied.

A method introduced in the late 80's by Omologo [4] may be considered a precursor of our method, because it uses the values of LSFs in the previous frame as starting point in an iterative algorithm. However this algorithm is completely different of ours and while it has a good average behavior, it is uncontrolled with respect to worst case performance.

There is no absolute ranking of the cited methods, one method outperforming the others in some particular situations. Therefore we confine our comparison to the case of real-time speech coders, as those used in mobile telephony. Kabal's algorithm seems the most appropriate to this field; its execution time is approximately constant regardless of the polynomials and the corresponding code is small. The Wu-Chen algorithm is faster when the LSPs should be computed with high accuracy, which is not the case here; moreover, the procedure being partly iterative, the worst case behavior may be far enough of the average.

The recent ITU-T standard G.729 [2] implements the Kabal-Ramachandran algorithm, for the case $m = 5$. The evaluation grid has 60 points (equally spaced in frequency); after bracketing a root, 4 bisections are performed. These values lead to a total worst case of 109 polynomial evaluations. (P_r is also evaluated in the roots of Q_r and vice-versa, excepting for the first computed root.)

2 USING PREDICTIONS FOR FAST ROOT BRACKETING

The use of predicted LSFs for the fast computation of the current ones was not exploited up to now. It is well recognized that there is a correlation between the LSFs belonging to successive frames of a sampled speech signal. This correlation is actually used in the quantization stage. There are several methods proposed to this purpose, among which the most explicit is to design a linear prediction model used to approximate the current (computed) LSFs set ω_i, $i = 1 : n$, with a set $\hat{\omega}_i$ and to quantize $\omega_i - \hat{\omega}_i$. For example, in G.729, two fourth order MA predictors are used in order to provide a better LSF approximation.

Our method uses the predicted LSFs for the computation of the current LSFs, with the goal of reducing the complexity of root searching.

There are several tracks that could lead from the approximations to the exact LSFs. We have chosen a solution which is near in flavor to the Kabal-Ramachandran algorithm. That is, working on the same grid covering the interval $(-1, 1)$, we try to find a grid interval which brackets the LSP, starting the search in the predicted LSP and using additional information offered by the sign and the slope of the polynomial around the prediction. Let us denote x_i the LSP to be computed and $\hat{x}_i$ the predicted LSP.

Let us suppose that we already have computed the first $i-1$ LSPs and we want to compute x_i. The interval $(-1, 1)$ is covered with a grid of points g_l, $l = 1 : L$, conventionally ordered from 1 to -1 (in G.729, $L = 60$). Due to the ordering of the LSPs set, there is an index k_m such that $x_i < g_{k_m}$, the larger for which $x_{i-1} \leq g_{k_m}$; in other words, g_{k_m} is the first grid point at the right of x_{i-1}; we will use the notation $g_{k_m} = x_{i-1}\rfloor$. Let suppose that x_i is a root of P_r and that $P_r(x_{i-1}) > 0$; this case is immediately extended for the other polynomial and for opposite sign. The prediction $\hat{x}_i$ of the current LSP is available through the largest grid index $\hat{k}$ such that $\hat{x}_i \leq g_{\hat{k}} = \hat{x}_i\rfloor$. To complete the notation, let k be the grid index for which $g_k = x_i\rfloor$, i.e. the LSP x_i is within the interval $(g_{k+1}, g_k]$. Our problem is to find k starting from $\hat{k}$.

The possible locations of the prediction are presented in figure 1, where the notations $s = g_{\hat{k}}$, $s_1 = g_{\hat{k}+1}$ are used. First of all, if $\hat{k} \leq k_m$, i.e. the prediction is at the right of the last computed LSP, we reassign $\hat{k} = k_m + 2$. Then, the assumption is made that the prediction is never so bad such that $g_{\hat{k}} < x_{i+4}$ (the experiments largely confirmed this assumption). The interval (x_{i+4}, x_{i-1}) is now divided in four categories, marked by vertical stripes in figure 1.

The algorithm we propose is outlined in figure 2.

The first classification is made upon the sign of $P_r(s)$. If the sign is negative, i.e. the prediction is in stripe C, the root is already bracketed and further search will be oriented to the right of s. The interval conataining the root is found by going backwards from $l = \hat{k} - 1$ down to k_m, with step δ_C (whose value will be chosen later), until finding a positive value. (Of course, if $\delta_C > 1$, further bisections are needed to obtain one grid interval containing the root.)

If $P_r(s)$ is positive, then the polynomial is evaluated in the next grid point $s_1 = g_{\hat{k}+1}$. Now, if $P_r(s_1)$ is negative, we fall in stripe B, which is the best case: the LSP is bracketed with two polynomial evaluations !

If $P_r(s_1)$ is also positive, the algorithm we propose is the following. If $P_r(s_1) < P_r(s)$, the corresponding stripes are

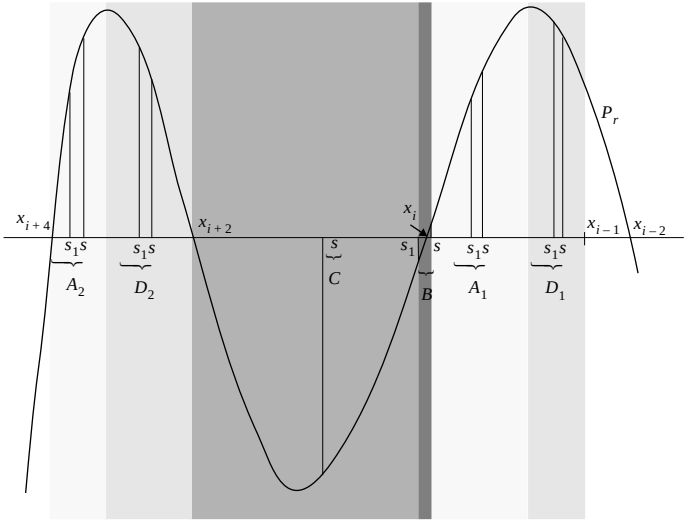

Figure 1: Possible locations of the current predicted LSP.

A_1 and A_2. In A_1 the strategy is exactly the same of the Kabal-Ramachandran algorithm, since there is no other information available about the root; that is, the polynomial is evaluated on successive grid points starting from $\hat{k} + 2$ until finding a sign change. The case A_2 must be avoided, since it is indistinguishable of A_1; the solution we adopted is to choose a threshold value Δ and, if $\hat{k} - k_m > \Delta$, to recompute the "prediction" with $\hat{k} = k_m + \Delta/2$. The choice of Δ is clearly a compromise: a large value will make less sure the avoidance of the region A_2, while a small value will possibly modify good predictions when $x_{i-1} - x_i$ is large.

If $P_r(s_1) \geq P_r(s)$ the corresponding stripes are D_1 and D_2, with the mention that D_1 might not exist, depending on the position of x_{i-1}, which is a root of Q_r, with respect to the maximum of P_r in the interval $[x_i, x_{i-2}]$. In this case, we chose to reset $\hat{k}$ to k_m and to proceed like in case A.

We will name PK the described algorithm (standing for predictive Kabal-Ramachandran). It is clear that the algorithm is not dependent on all the details of the basic Kabal algorithm, but only on the existence of a prediction of the LSPs and the limitation of the search on some grid points. However, we will report results comparing to the original Kabal-Ramachandran algorithm, since this is largely used currently.

3 TUNING OF PARAMETERS

The experiments were performed on the TIMIT database, the test part, on over 500,000 speech frames processed with the G.729 floating point coder (single precision), at which we have attached our routine for predictive LSP computation.

The quality of the prediction of the G.729 coder is indicated by the percentages of predictions falling in the four categories A, B, C, D from figure 1. The first line of table 1 is showing the result without modifying the prediction. For the second line, $\hat{k}$ is increased to $k_m + 2$ if the prediction is at the right of D_1. We notice that the categories that raise difficulties, i.e. D and especially A_2, have low frequencies, while over a quarter of the predictions fall into stripe B, where the LSP is bracketed with two evaluations.

Choice of Δ. The threshold Δ is introduced to avoid the location of the prediction in stripe A_2. Measuring, for all the speech frames, the distance $x_{i+4}\rfloor - x_{i-1}\rfloor$ (in number of

1. For the root x_i of P_r
   1.0 Denote $x_{i-1}\rfloor$ the nearest right grid point of x_{i-1} and s the nearest right grid point of the root prediction $\hat{x}_i$
   If $s \geq x_{i-1}\rfloor$ (prediction right of latest computed root), then $s \leftarrow x_{i-1}\rfloor + 1$
   Denote s_1 the next grid point at left of s
   Suppose that $P_r(x_{i-1}) > 0$ (if not, reverse all the inequalities involving P_r)
   1.1 Evaluate the polynomial in s: $P_r(s) = horner(P_r, s)$
   1.2 If $P_r(s)$ and $P_r(x_{i-1})$ have the same sign and the prediction s of the current root is far-off of previous root
   Then move s to the right
   (If $s - x_{i-1}\rfloor > \Delta$ then $s = x_{i-1}\rfloor + \frac{1}{2}\Delta$)
   1.3 If $P_r(s)$ and $P_r(x_{i-1})$ have different signs
   CASE C
   Move on the grid from s to the right with step δ_C $(s \leftarrow s - \delta_C)$
   Evaluate at each grid point the polynomial $P_r(s) = horner(P_r, s)$
   Until the root is bracketed (when $P_r(s) > 0$)
   Else
   Evaluate the polynomial in s_1: $P_r(s_1) = horner(P_r, s_1)$
   If $P_r(s_1) < 0 < P_r(s)$
   CASE B
   The root is bracketed with no further evaluations !
   Else if $0 < P_r(s_1) < P_r(s)$
   CASE A
   Move on the grid from s to the left with step δ_A $(s_1 \leftarrow s_1 + \delta_A)$
   Evaluate at each grid point the polynomial $P_r(s_1) = horner(P_r, s_1)$
   Until the root is bracketed $(P_r(s_1) < 0)$
   Else (i.e. $0 < P_r(s) < P_r(s_1)$)
   CASE D
   Reassign $s \leftarrow x_{i-1}\rfloor + 1$
   Move on the grid from s to the left with step δ_D $(s_1 \leftarrow s_1 + \delta_D)$
   Evaluate at each grid point the polynomial $P_r(s_1) = horner(P_r, s_1)$
   Until the root is bracketed $(P_r(s_1) < 0)$

Figure 2: Description of fast bracketing in the predictive LSP computation algorithm. The routine to compute the value of a polynomial is generically named *horner*. The notation s and s_1 are used simultaneously for a grid point and for the corresponding index.

grid intervals), we have found this distance smaller than 12 in less than 10^{-5}. Combined with the even lower apparition of predictions in stripe A_2 (see again table 1), the choice $\Delta = 12$ ensures that the probability to obtain both these events is negligible. The third line of table 1 shows that introducing Δ eliminates the case A_2 and does not modify significantly the relative frequencies of other cases.

Choice of δ_C. It is helpful to see how large the values of $d = \hat{k} - k$ may be when the prediction falls in stripe C (after applying the "Δ rule"); we found out that although d is usually very small ($d = 1$ in 50% of cases), it may take large values, going up to 19.

Therefore, in the algorithm in figure 1, we chose $\delta_C = 2$, significantly improving on the worst case with respect to the choice $\delta_C = 1$. We denote PK_2 the corresponding algorithm.

4 PERFORMANCE OF THE PREDICTIVE LSP ALGORITHM

The performance of our algorithm is reported in number of polynomial evaluations ("horners") N_{horner} required for the computation of a LSP set. This measure does not take into account the way the polynomial is evaluated and may be as well suitable for the straight Horner rule as for the Chebyshev version. Agreeing that four bisections are needed to refine a bracketed root, it is obvious that the minimum possible value of N_{horner} is 60, in the case where all LSPs of a speech frame are predicted with high accuracy (all predictions are in stripe B); a worst case is difficult to evaluate; for example, if all predictions are at the right of the previous computed LSP, the number of evaluations will be the same as in Kabal's algorithm.

The first plot is in the left side of figure 3, indicating the number of occurrences of N_{horner} on the whole test data for the algorithm PK_2. The average value of N_{horner} is 70.68. The worst value is 99. The frequencies of the number of occurrences on length 5 intervals are shown in table 2. It results that the frequency of a N_{horner} greater than 90 is very small and that the event $N_{horner} > 80$ has a frequency less than 4%.

To compare with Kabal-Ramachandran, we show in the same figure 3 the number of occurrences of N_{horner} for this algorithm. The average value is 104.75 polynomial evaluations and the worst case 109, as we already mentioned.

5 CONTROLLING THE WORST CASE BEHAVIOR

In a real time implementation, the worst case of the execution time is the best measure of the performance. The

	$< D_1$	D_1	A_1	B	C	D_2	A_2
$\hat{k}$ not modified	1.44	1.26	32.36	24.10	40.81	0.038	0.00025
$\hat{k} \leftarrow \max(\hat{k}, k_m + 2)$	0	0.41	30.27	26.25	43.04	0.038	0.00025
$\hat{k} \leftarrow k_m + \Delta/2$ if $\hat{k} > k_m + \Delta \quad (\Delta = 12)$	0	0.49	30.30	26.14	43.05	0.020	0

Table 1: Frequencies of prediction categories.

N_{horner}	60–65	66-70	71–75	76–80	81–85	86–90	91–95	96–99
Frequency of occurrence (%)	9.24	46.25	30.83	9.76	3.14	0.70	0.075	0.0046

Table 2: Frequencies of number of occurrences of N_{horner} for algorithm PK_2.

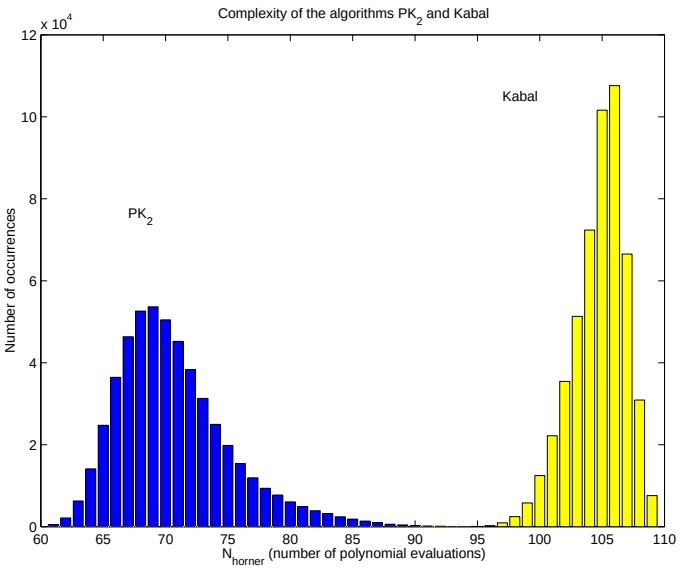

Figure 3: Frequencies of the number of polynomial evaluations for algorithms PK_2 (left side) and Kabal-Ramachandran (right side).

method presented up to now performs very well in average, giving a significant gain over the Kabal-Ramachandran method, but the worst case time is only about 10% better. There is a simple extension to cope with this difficulty, by imposing a maximum number of horners N_{max} as upper limit for the execution of the algorithm. During the execution of the algorithm, we monitor the number of polynomial evaluations. The algorithm is reorganized in two steps:

1. Each root is only bracketed in one grid interval using the algorithm described in figure 1. At the end of this step, the number of polynomial evaluations has the value N_1.

2. The resulted intervals are bisected to refine the root. The number of bisections per root N_b that may be performed within the limit N_{max} is $N_b = \lfloor (N_{max} - N_1)/n \rfloor$.

Of course, the proposed strategy may affect the accuracy of some computed roots, so N_{max} should be chosen carefully. Let us denote $PK_2(N_{max})$ the algorithm obtained from PK_2 using the technique described above.

Looking again at table 2, we remind that $N_{horner} > 80$ appears only in 4% of the cases. Thus, using $N_{max} = 80$ will affect the accuracy of a very small number of LSP sets.

Algorithm	average	2-4 dB	> 4 dB
PK_2	1.340	11.79%	0.457%
$PK_2(80)$	1.341	11.81%	0.458%
$PK_2(75)$	1.341	11.81%	0.458%

Table 3: Spectral distortion in predictive LSP computation.

For the algorithm $PK_2(80)$, the average of N_{horner} is 70.35, slightly modified with respect to algorithm PK_2. Table 3 shows that the average spectral distortion is practically unchanged and the same conclusion applies for the number of outliers between 2 and 4dB or over 4dB. A similar behavior has the algorithm $PK_2(75)$, for which the average number of horners is 69.3. These results show that the idea of limiting the number of polynomial evaluations is successful.

References

[1] F. Itakura. Line Spectrum Reprezentation of Linear Predictor Coefficients of Speech Signals. *J. Acoust. Soc. Amer.*, 57(S35(A)), 1975.

[2] ITU-T. Recommendation G.729 — Coding of Speech at 8 Kbits/s Using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP), March 1996.

[3] P. Kabal and R.P. Ramachandran. Computation of Line Spectral Frequencies Using Chebyshev Polynomials. *IEEE Trans. Acoust., Speech, Signal Proc.*, 34(6):1419–1426, December 1986.

[4] M. Omologo. The computation and some spectral considerations on line spectrum pairs (LSP). In *European Conference on Speech Communication and Technology, Eurospeech*, volume 2, pages 352–355, Paris, France, 1989.

[5] C.H. Wu and J.H. Chen. A Novel Two-Level Method for the Computation of the LSP Frequencies Using a Decimation-in-Degree Algorithm. *IEEE Trans. Speech Audio Proc.*, 5(2):106–115, March 1997.