

TABLE I
DETERMINATION OF ENERGY LEVEL THRESHOLDS FROM KNOWN VOICING AND NOISE LEVELS

$K_4 = v - 8$ dB	Above this is definitely voicing
$K_2 = v - 20$ dB	Minimum for any voiced sound
$K_3 = K_5 + 10$ dB	Above most noise peaks
$K_1 = K_5 + 5$ dB	High noise peak

TABLE II
VALUES FOR THE TIME BASED THRESHOLDS

T1	40 ms	Energy risetime—precision of Word Boundary starting point
T2	40 ms	Energy falltime—precision of Word Boundary ending point
T3	75 ms	Minimum valid pulse width, for Pulse Detection.
NF	75 ms	Minimum valid pulse length, for Pulse Rejection. Should be $\geq T3$.
NSEP	150 ms	Pulses separated by more than this may be separate utterances.
MAXLENGTH	1 sec	Pulses separated by more than this are definitely separate utterances.

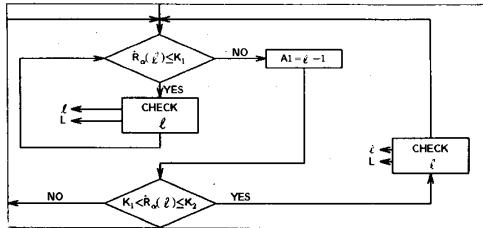


Fig. 1. Corrected part of the energy pulse detector, from the first segment of Fig. 5(a) of Lamel *et al.*'s paper.¹

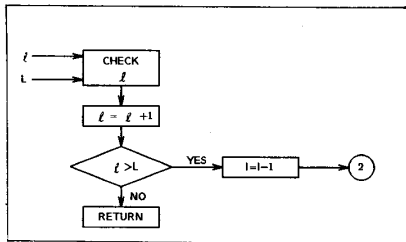


Fig. 2. $I = I - 1$ has been inserted. From Fig. 5(a), third segment, Lamel *et al.*¹

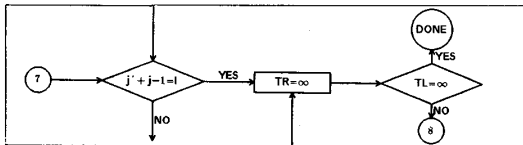


Fig. 3. The prime has been removed from "I" in the left decision. Corrected from Lamel *et al.*, Fig. 5(b), third segment.

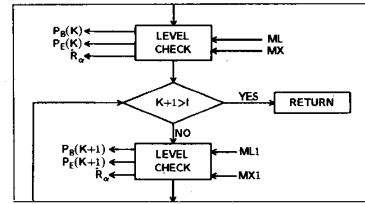


Fig. 4. The lower LEVEL CHECK procedure's arguments are indexed by $K + 1$, corrected from Lamel *et al.*, Fig. 5(b), fourth segment.

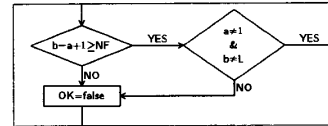


Fig. 5. The "NO" result on the right side decision is added to the last segment of Lamel *et al.*, Fig. 5(b).

REFERENCES

[1] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*. New York: Cambridge, 1988, p. 480.

Adaptive Polynomial Factorization by Coefficient Matching

David Starer and Arye Nehorai

Abstract—A new polynomial factorization algorithm is presented which updates all roots simultaneously and efficiently in response to coefficient perturbations. The algorithm requires approximately $2n^2$ complex floating point operations to update all roots of an n th order polynomial. Close to the true root vector, the algorithm's convergence rate is quadratic. The root update only requires the solution of two sets of structured linear equations and a convolution. The algorithm can be used to track the roots of time-varying polynomials which is useful for applications in adaptive signal processing.

I. INTRODUCTION

Polynomial factorization is often required by signal processing and system identification algorithms. Examples include speech processing, frequency estimation of multiple complex exponential signals in noise, and source localization using uniform linear array sensors. In many cases, the polynomials are time varying and their coefficients can be tracked using well-established algorithms such as the recursive prediction error (RPE) method [1]. However, as will be explained later, tracking the movements of the roots cannot be accomplished efficiently with conventional factorization methods.

Manuscript received August 14, 1989; revised March 12, 1990. This work was supported by the Air Force Office of Scientific Research under Grant AFOSR-88-0080.

The authors are with the Department of Electrical Engineering, Yale University, New Haven, CT 06520.
IEEE Log Number 9041109.

Conventional polynomial factorization algorithms (see, e.g., [4]) usually seek parameter values for which the polynomial evaluates to zero. Once such a value (or root) is located, the order of the polynomial is reduced using deflation. These operations are repeated until all roots have been estimated. This procedure has several disadvantages which make it unsuitable for implementation in adaptive algorithms: a) individual roots must be estimated iteratively to maximum accuracy one at a time, b) inaccuracies can accumulate when using deflation, and c) time-varying polynomial coefficients cannot be handled efficiently, i.e., if the coefficients change slightly, the entire factorization procedure must be restarted.

This correspondence¹ presents a new efficient algorithm which overcomes the above difficulties by using a different approach based on coefficient matching rather than on explicit zero finding. It estimates all roots simultaneously so that deflation is not needed. This eliminates the necessity for estimating individual roots to maximum accuracy at each iteration. The method is suitable for tracking the roots of time-varying polynomials.

For certain polynomials, for example, those associated with autoregressive (AR) and autoregressive moving-average (ARMA) processes, on-line methods for tracking roots directly from the data have been proposed recently (see, e.g., [2] and the references therein). However, in many cases, the time-varying coefficients themselves constitute the data, and it is necessary to track the associated roots. This problem was solved by the two-stage algorithm proposed in [2, appendix] using a steepest descent approach which often exhibits slow convergence. The algorithm derived here presents a solution to this problem, utilizing a Gauss-Newton method which exhibits more rapid convergence.

In the adaptive signal processing context, root estimation can be regarded as a two-stage procedure as shown in Fig. 1. In the first stage, with the reception of every data sample, the coefficient vector is updated. For example, for ARMA signals, this can be accomplished with the RPE algorithm. In the second stage, also with the reception of every data sample, the root vector is updated. This paper is concerned with obtaining an efficient algorithm for use in the second stage of the root estimation procedure.

A. Problem Statement

Given a vector a of polynomial coefficients

$$a = [a_1, a_2, \dots, a_n]^T \quad (1.1)$$

it is required to compute the vector λ of polynomial roots

$$\lambda = [\lambda_1, \lambda_2, \dots, \lambda_n]^T \quad (1.2)$$

such that

$$f(z) = \sum_{k=0}^n a_k z^{-k} = \prod_{k=1}^n (1 - \lambda_k z^{-1}); \quad a_0 = 1 \quad (1.3)$$

for arbitrary $z \in \mathbb{C}$. It is assumed that $\lambda_i \neq \lambda_j$ for all $i \neq j$. The vectors a and λ can have complex as well as real elements.

II. FACTORIZATION ALGORITHM

In contrast to conventional factorization algorithms which search for values of z which cause the polynomial to evaluate to zero, the proposed algorithm minimizes a cost function which measures the squared error between the given polynomial coefficients and those corresponding to the current estimate of the roots (see also [2, appendix]). The cost function is

$$V(\lambda) = \varepsilon^H(\lambda)\varepsilon(\lambda) \quad (2.1a)$$

where the superscript H is used to denote complex conjugate transpose and the error is

$$\varepsilon(\lambda) = a - \hat{a}(\lambda) \quad (2.1b)$$

¹See also ICASSP-89 [3] where portions of this correspondence first appeared.

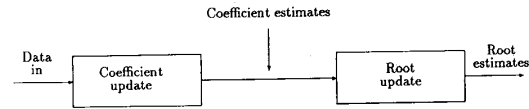


Fig. 1. Adaptive two-stage root tracking.

where $\hat{a}(\lambda)$ is the vector of coefficients derived from the current estimate of the root vector λ . Note that the cost function (2.1a) is a quadratic function of \hat{a} , but it is a nonlinear function of λ . Thus, as a function of the estimated coefficients, the cost function exhibits only one minimum, and this corresponds to the true coefficients. Since the roots of a polynomial are unique, any estimated root vector giving rise to the true coefficient vector must contain only the true roots. However, these roots can be arranged in any order. Thus, as a function of the root vector, the cost function exhibits many minima but each one of these minima is a valid arrangement of the true roots. Therefore, convergence of the algorithm is guaranteed.

Using the Newton algorithm for the minimization of (2.1a), the expression for the $(k+1)$ st iteration is

$$\hat{\lambda}^{(k+1)} = \hat{\lambda}^{(k)} - \left[\left[\frac{\partial^2 V(\lambda)}{\partial \lambda \partial \lambda^H} \right]^{-1} \frac{\partial V(\lambda)}{\partial \lambda} \right]_{\lambda = \hat{\lambda}^{(k)}} \quad (2.2)$$

where differentiation with respect to a complex variable is performed in the sense described by Brandwood [5]. Differentiation of the cost function gives the following expressions for the gradient and the Hessian:

$$\frac{\partial V(\lambda)}{\partial \lambda} = 2 \frac{\partial \varepsilon^H}{\partial \lambda} \varepsilon \quad (2.3)$$

$$\frac{\partial^2 V(\lambda)}{\partial \lambda \partial \lambda^H} = 2 \frac{\partial \varepsilon^H}{\partial \lambda} \frac{\partial \varepsilon}{\partial \lambda^H} + \varepsilon''(\lambda)\varepsilon(\lambda). \quad (2.4)$$

The first term of the Hessian is positive definite for parameter vectors close to the true value. The second term represents a tensor product which is proportional to the error ε . This tends to zero as the estimate approaches the true parameter value, and can therefore be neglected. This is a standard approximation used in Gauss-Newton methods (see, e.g., [1]) and ensures that the Hessian remains positive definite. With this approximation, the recursion (2.2) becomes

$$\hat{\lambda}^{(k+1)} = \hat{\lambda}^{(k)} - \left[\left[\frac{\partial \varepsilon(\lambda)}{\partial \lambda^H} \right]^{-1} \varepsilon(\lambda) \right]_{\lambda = \hat{\lambda}^{(k)}} \quad (2.5)$$

From (2.1b), it is clear that

$$\frac{\partial \varepsilon}{\partial \lambda^H} = - \frac{\partial \hat{a}(\lambda)}{\partial \lambda^H}. \quad (2.6)$$

It has been shown [6] that for polynomials with simple roots

$$\frac{\partial \hat{a}(\lambda)}{\partial \lambda^H} = -HV \quad (2.7)$$

where H is a Hankel matrix of polynomial coefficients and V is a Vandermonde matrix of polynomial roots as defined in [6]. For convenience in applying existing efficient algorithms (mentioned later) for this update, it is useful to reverse the columns of H and the rows of V . This does not affect the result, but leads to the equivalent gradient expression

$$\frac{\partial \hat{a}(\lambda)}{\partial \lambda^H} = -TA \quad (2.8a)$$

where T is the lower triangular Toeplitz matrix defined as

TABLE I
REAL PART OF ROOT ESTIMATES FOR THE POLYNOMIAL OF EXAMPLE 1

Iteration	Real Part of Root Estimates				
0	0.30901699	-0.80901699	-0.80901699	0.30901699	1.00000000
1	0.95013156	0.18986844	0.18986844	0.95013156	1.42000000
2	0.94760482	0.17717863	0.17717863	0.94760482	1.45043310
3	0.94373201	0.13599518	0.13599518	0.94373201	1.54054563
4	0.96945258	0.04650992	0.04650992	0.96945258	1.66807500
5	1.00167648	-0.00134992	-0.00134992	1.00167648	1.69934688
6	9.99999941	0.00000077	0.00000077	9.99999941	1.69999965
7	1.00000000	0.00000000	0.00000000	1.00000000	1.70000000
8	1.00000000	0.00000000	0.00000000	1.00000000	1.70000000
9	1.00000000	0.00000000	0.00000000	1.00000000	1.70000000
10	1.00000000	0.00000000	0.00000000	1.00000000	1.70000000

$$T = \begin{bmatrix} 1 & & & 0 \\ a_1 & 1 & & \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & \cdots & a_1 & 1 \end{bmatrix} \quad (2.8b)$$

and Λ is the Vandermonde matrix defined as

$$\Lambda = \begin{bmatrix} 1 & \cdots & 1 \\ \lambda_1 & \cdots & \lambda_n \\ \vdots & & \vdots \\ \lambda_1^{n-1} & \cdots & \lambda_n^{n-1} \end{bmatrix} \quad (2.8c)$$

Substitution of the gradient (2.8a) into (2.5) gives the final expression for the Gauss-Newton root estimate iteration

$$\hat{\lambda}^{(k+1)} = \hat{\lambda}^{(k)} - [T\Lambda]^{-1} \varepsilon. \quad (2.9)$$

The structure of the matrices T and Λ enables efficient methods to be employed to perform the recursion (2.9) as follows. Let the "update" be defined as

$$x = [T\Lambda]^{-1} \varepsilon. \quad (2.10a)$$

This can be found by solving for y in

$$Ty = \varepsilon \quad (2.10b)$$

followed by solving for x in

$$\Lambda x = y. \quad (2.10c)$$

Since T is lower triangular, y can be calculated using a simple forward substitution algorithm requiring $n^2/2$ flops. (Note, however, that more efficient algorithms are possible since solving this Toeplitz system amounts to polynomial division and this can be done in $O(n \log n)$ operations by applying Newton's method to the division of formal power series [7, pp. 515-520]). The solution x of the Vandermonde system can be computed using the Björck and Pereyra method [8, algorithm 5.6-2] requiring n^2 flops. Mapping the roots to the coefficients is required in order to construct T and for calculation of ε . The mapping can be performed by a convolution requiring $n^2/2$ flops (see, e.g., [3]). Thus, the total number of computations required by this algorithm is $2n^2$ flops.

The algorithm can be operated in two different "modes." These are: a) The off-line mode where the algorithm is iterated until convergence. This is analogous to conventional polynomial factorization. b) The on-line mode, where the algorithm is used to track

time-varying roots. In the on-line mode, it has generally been found satisfactory to perform only one root vector update per coefficient update. This is true because the proposed factorization algorithm typically converges considerably faster than coefficient estimators.

A. Limitations

The algorithm requires the matrices T and Λ to be nonsingular. This is assured for T , since it is clear that $\det T = 1$. However, Λ can be singular if the assumption $\lambda_j \neq \lambda_k$ is not satisfied for all j and k . Thus, in its present form, the algorithm cannot be used to factorize polynomials with multiple roots.

If complex roots are sought for a polynomial with real coefficients, the initial root estimates must have nonzero imaginary parts since the algorithm cannot generate complex estimates from purely real initial values.

III. EXAMPLES

Example 1: To demonstrate its behavior in the off-line mode, the algorithm was used to find the roots of the polynomial

$$f(z) = 1 - 3.7z^{-1} + 7.4z^{-2} - 10.8z^{-3} + 10.8z^{-4} - 6.8z^{-5} \quad (3.1)$$

with which Conte and de Boor [4, p. 126] tested factorization algorithms. The exact roots of this polynomial are $1 \pm i$, $\pm \sqrt{2}i$, and 1.7 . The algorithm was initialized with roots placed uniformly around the unit circle in the complex plane, and 14-digit precision was used. The results obtained are given in Tables I and II, showing that accurate estimates are found from inaccurate initial conditions in approximately seven iterations in this case.

Example 2: To demonstrate the tracking performance of the algorithm in the on-line mode, it was used to find the roots of a tenth-order time-varying polynomial. The polynomial had stationary roots at $\pm 0.6 \pm 0.4i$ and $\pm 0.2 \pm 0.8i$, and a pair of time-varying roots located at $0.7 \exp(\pm i\phi(t))$ where $\phi(t)$ was a time-varying parameter. The algorithm was initialized with the true roots. One root vector update was performed per coefficient update (or time sample).

Fig. 2 shows the results obtained when $\phi(t)$ varied sinusoidally. The dashed line indicates the true value of $\phi(t)$ and the asterisks represent the estimated value. It is apparent in this case that a single update is sufficient to track the roots.

IV. CONCLUSION

This correspondence has presented a new polynomial factorization algorithm suitable for use in off-line and on-line modes, but is particularly suitable for adaptive signal processing applications where it is required to track the movements of roots. For this purpose, factorization is regarded as a coefficient matching problem: The polynomial roots constitute the parameter vector and the coef-

TABLE II
IMAGINARY PART OF ROOT ESTIMATES FOR THE POLYNOMIAL OF EXAMPLE 1

Iteration	Imaginary Part of Root Estimates				
0	0.95105652	0.58778525	-0.58778525	-0.95105651	0.00000000
1	0.92261343	6.24810247	-6.24810247	-0.92261343	0.00000000
2	0.92985047	3.25777385	-3.25777385	-0.92985047	0.00000000
3	0.95607105	1.89465271	-1.89465271	-0.95607105	0.00000000
4	1.00765391	1.43784684	-1.43784684	-1.00765391	0.00000000
5	1.00145623	1.41286153	-1.41286153	-1.00145623	0.00000000
6	1.00000346	1.41421016	-1.41421016	-1.00000346	0.00000000
7	1.00000000	1.41421356	-1.41421356	-1.00000000	0.00000000
8	1.00000000	1.41421356	-1.41421356	-1.00000000	0.00000000
9	1.00000000	1.41421356	-1.41421356	-1.00000000	0.00000000
10	1.00000000	1.41421356	-1.41421356	-1.00000000	0.00000000

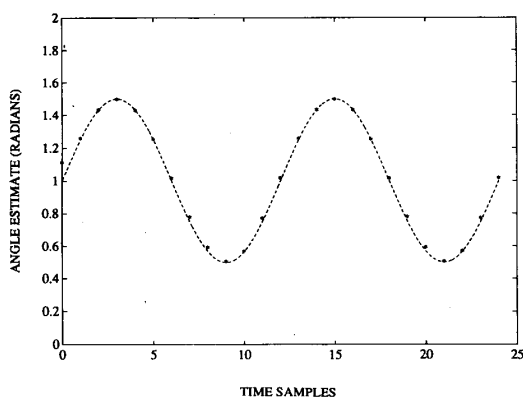


Fig. 2. Tracking the nonstationary roots of the tenth-order polynomial of Example 2.

ficients are treated as data to which a parametric model is to be fitted. This is unlike conventional factorization algorithms which explicitly search for roots of the given polynomial (i.e., values of z such that $f(z) = 0$). The algorithm updates all roots simultaneously, thus avoiding deflation.

Unlike available direct adaptive pole estimation algorithms [2], the method is not restricted to estimation of the roots of ARMA-like processes, but can be used in any situation where on-line estimates of polynomial coefficients are available, for example, in direction of arrival estimation using uniform linear arrays. However, for ARMA-like models, the direct algorithm proposed in [2] is more computationally efficient.

The new algorithm converges more rapidly than the steepest descent algorithm described in [2, appendix]. Close to the true root vector, the proposed algorithm's convergence rate is quadratic, and it updates the entire root vector simultaneously in $2n^2$ complex flops.

The root estimates can be updated by performing a convolution to compute coefficient estimates from current root estimates, and by solving two sets of linear equations. The method is therefore attractive, for example, for hardware implementation. The linear equations derived are highly structured, so efficient techniques can be employed to solve them.

REFERENCES

- [1] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification*. Cambridge, MA: M.I.T. Press, 1983.
- [2] A. Nehorai and D. Starer, "Adaptive pole estimation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, no. 5, pp. 825-838, May 1990.
- [3] D. Starer and A. Nehorai, "Polynomial factorization algorithms for adaptive root estimation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing* (Glasgow, Scotland), May 1989, pp. 1158-1161.
- [4] S. D. Conte and C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, third ed. New York: McGraw-Hill, 1980.
- [5] D. H. Brandwood, "A complex gradient operator and its applications in adaptive array theory," *Proc. Inst. Elec. Eng.*, vol. 130, pts. F and H, pp. 11-15, Feb. 1983.
- [6] A. Nehorai and B. Porat, "Adaptive comb filtering for harmonic signal enhancement," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 1124-1138, Oct. 1986.
- [7] P. Henrici, "Fast Fourier methods in computational analysis," *SIAM Rev.*, vol. 21, pp. 481-527, Oct. 1979.
- [8] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: Johns Hopkins University Press, 1983.

A Relationship Between the Recursive Least Squares Update and Homotopy Continuation Methods

Virginia L. Stonick and S. T. Alexander

Abstract—This correspondence presents an alternate derivation of the general recursive least squares (RLS) algorithm using homotopy approaches. Homotopy continuation methods are briefly described, and a simple iterative update for path following is derived. A homotopy function is then constructed using the RLS filtering criteria. The equivalence between the iterative update for following the path defined by this homotopy function and the general RLS update is derived. Finally, an intuitive explanation of this result is presented.

Manuscript received November 9, 1989; revised March 28, 1990. This work was supported in part by the Bell Communications Research Graduate Fellowship Program and by the National Science Foundation under Grant MIP-8552571.

V. L. Stonick was with the Electrical and Computer Engineering Department, North Carolina State University, Raleigh, NC 27695-7911. She is now with the Electrical and Computer Engineering Department, Carnegie-Mellon University, Pittsburgh, PA 15213.

S. T. Alexander is with the Electrical and Computer Engineering Department, North Carolina State University, Raleigh, NC 27695-7911.

IEEE Log Number 9041118.