Polynomial Root Finding

Markus Lang, Bernhard-Christian Frenzel

August 10, 1994

Abstract

Finding polynomial roots rapidly and accurately is an important problem in many areas of signal processing. We present a program which is superior in speed and accuracy to the best methods to our knowledge, i.e., Jenkins/Traub program and the eigenvalue method. Based on this we give a simple approach to improve the accuracy for spectral factorization in the case there are double roots on the unit circle.

EDICS: SPL 4.1

Contact Address:
Markus Lang
Electrical and Computer Engineering - MS 366
Rice University, Houston, TX-77251-1892
Phone: (713) 527-8101 x3569
FAX: (713) 524-5237
email: lang@jazz.rice.edu

List of Figures

1 Introduction

Finding polynomial roots rapidly *and* accurately is an important problem in various areas of signal processing such as spectral factorization (filter and wavelet design, spectral estimation), phase unwrapping, forming a cascade of lower order systems, educational purposes, etc. [1, 2, 8, 10]. There exists a large number of different methods for finding all polynomial roots either iteratively or simultaneously. Most of them yield accurate results only for small degrees or can treat only special polynomials, e.g., polynomial with real roots.

One of the two best general purpose root finders is the Jenkins/Traub method [4]. It works with the polynomial itself. The second method computes the eigenvalues of the companion matrix with high accuracy using the EISPACK routines [9].

We present a method for finding all polynomial roots of an arbitrary complex valued polynomial. It basically consists of a combination of Muller's and Newton's method [7].[1]

In Sec. 2 we describe the basic structure of our program. The efficiency and reliability of all three methods are compared in Sec. 3 where we use several test polynomials with known roots. We discuss how to deal with double roots on the unit circle in the case of spectral factorization in Sec. 4. The main results are summarized in Sec. 5.

A much more detailed description of the procedure and the evaluation of its performance can be found in [6].

2 Description of the Procedure

We consider a complex valued polynomial

$$P(x) = \sum_{\nu=0}^{n} p_\nu x^\nu = p_n \prod_{\nu=1}^{n} (x - x_\nu), \tag{1}$$

of degree n. The problem we address is that of finding all n roots x_ν of $P(x)$ as accurately and fast as possible. We work with a given and fixed computation accuracy, i.e., IEEE-P754-floating point standard (accuracy $\approx 2.2 \cdot 10^{-16}$). The relative accuracy of the resulting roots must be compared to this number. We have chosen a combination of Muller's and Newton's method since both of them can be used to find complex roots.

In the first step the coefficients are formally checked, e.g., possible roots at zero are determined and deflated. In the case of a first or second order polynomial the well-known explicit formulae are used. Only for degree $n > 2$ do we execute our iterative procedure. We use Muller's method to compute an estimate for a root of the actual, deflated polynomial which contains all roots of $P(x)$ except the roots found up to the actual iteration step (k). Muller's method [7] extends the idea of the secant method which works with a linear polynomial to a quadratic polynomial. One of its advantages is good convergence to a reasonable estimate of a root. Another is the

[1] A similar approach was chosen in [3, 11] that does not focus on high degree polynomials, does not extensively test for stability and accuracy, does not address the case of multiple roots on the unit circle and uses the secant instead of Newton's method.

ability to obtain complex roots even when initialized with real values. This is in contrast to other methods, e.g., Newton's method. The convergence speed is super linear (1.84 for single roots). General convergence has not been proven.

In the second step the estimate resulting from Muller's method is used as the initial value of Newton's method. It is simple and known to have at least quadratic convergence near the solution (for single roots). We use the original polynomial to avoid errors introduced by the deflation process. Once Newton's method has converged the resulting root is deflated (and possibly its complex conjugated). This procedure is repeated until the resulting polynomial is of degree two or one. An estimate for its root(s) can again be obtained by using the well-known explicit formula and then refined by Newton's method.

3 Performance of the New Algorithm

In this section we prove the efficiency of our algorithm. We point out that we have not attempted to construct a new algorithm with nice theoretical properties but have developed a tool which works reliably and fast in many practical applications. Thus we have made considerable efforts to avoid numerical problems like overflow and cycling while maintaining fast convergence. As Jenkins and Traub propose [5], we chose a lot of test polynomials with well-known roots that test programs for different weaknesses. A root finder is the better the smaller the difference between the correct roots and the determined ones. We use a normalized version of this criterion

$$e = \max_{\nu} \left| \frac{x_\nu - x_{min}}{x_\nu} \right|, \tag{2}$$

where x_{min} is the value computed by the root finder and x_ν is the corresponding exact value. This number is computed for every polynomial. Our program also computes an estimate $\tilde{e}$ for e.

Additionally, we determined the necessary CPU time on an HP Apollo workstation 9000/705. We compare the results of our program regarding speed and accuracy with two of the best generally available root finder programs to our knowledge. These are the Jenkins/Traub program [4] and the eigenvalue method based on EISPACK [9] in the version by MATLAB.

We do not show results of factoring actual transfer functions since we do not know the correct roots and thus cannot give an objective measure for the accuracy. However, we successfully computed the roots of many FIR filters. To give an example our program determined all roots of a degree 1000 FIR low-pass filter within 8.35s with an estimated error $\tilde{e} = 1.8 \cdot 10^{-16}$. The computed roots in the stopband which should have magnitude one have a maximum distance from the unit circle of $1.11 \cdot 10^{-16}$. That means they are exact within computer accuracy.

Test polynomials: Because of the limited space in this paper we present the results only for some of the test polynomials in [5]. We choose the polynomial

$$P_1(x) = \prod_{\nu=1-M}^{M-1} (x - \mathrm{e}^{\mathrm{j}\frac{\nu\pi}{2M}}) \cdot \prod_{\nu=M}^{3M} (x - 0.9 \cdot \mathrm{e}^{\mathrm{j}\frac{\nu\pi}{2M}}) \tag{3}$$

which has a root distribution similar to the transfer function of an FIR low-pass filter to check stability of the deflation, and the polynomial

$$P_2 = x^n - 1 \tag{4}$$

to check the performance of the programs for numerically well conditioned but high degree polynomials. For the second polynomial a good root finder should be able to determine the roots within computer accuracy.

For $P_1(x)$ and $n = 20 \ldots 200$ the accuracy of the Jenkins/Traub program is significantly smaller for degree 40 and the results are completely useless for higher degrees (Fig. 1(a)). The accuracy of our method is the best for all degrees and especially better by a factor of 1000 for $n = 200$ compared to the eigenvalue method. Regarding the accuracy, a comparison of the CPU time makes sense only between our and the eigenvalue method. Our method needs less time for all degrees (Fig. 1(b)). Especially for $n = 200$: it needs 2.11s compared to 96.4s.

As a second example, we present the results when working with the polynomial $P_2(x)$ in Fig. 2. Again, the accuracy of the Jenkins/Traub program drastically decreases for relatively small degrees ($n = 50$). It cannot be used for degrees $n > 60 \ldots 70$ (Fig. 2(a)). On the other hand our method yields the best results with near computer accuracy up to degree $n = 10000$. The eigenvalue method has slightly poorer accuracy but could be used only for degrees up to about 500. This is because of the necessary large memory to store the companion matrix $n^2 \cdot 8$Byte (double precision) which is $2 \cdot 10^6$Byte for $n = 500$ and $8 \cdot 10^8$Byte for $n = 10000$. The CPU time for our method is always the smallest as can be seen in Fig. 2(b). Especially for $n = 500$: it is 2.64s compared to 316.1s. In all examples above the estimate $\tilde{e}$ is close to the actual accuracy of our method.

4 Spectral Factorization

The problem of spectral factorization is to find all roots of a symmetric polynomial $H(x)$ (a polynomial where the existence of a root at x_ν implies the existence of a root at $(x_\nu^*)^{-1}$) with the additional property that all roots on the unit circle have even multiplicity. In the following we assume they have multiplicity two. After finding the roots of $H(x)$ one is interested in forming a minimum phase polynomial or in general a polynomial $P(x)$ of degree n such that

$$H(x) = P(x) \cdot x^n P^*((x^*)^{-1}) \tag{5}$$

holds.

As is well-known, the limiting accuracy for double roots is half the computer accuracy which obviously affects the roots on the unit circle. However, in the special case where we know their modulus it is possible to use a simple procedure to compute them up to computer accuracy. This can be done by the following procedure where we assume that all roots on the unit circle are well separated, which holds in all practical cases.

In the first step we compute all zeros of the original polynomial $H(x)$ and separate them into those lying inside, outside, and on the unit circle. To separate these three regions we chose an annulus with thickness 100 times the expected accuracy for which we use the estimate $\tilde{e}$ of our program. In the second step we compute the roots of $H'(x) = dH(x)/dx$. It has the same roots on the unit circle as the original polynomial $H(x)$ but with multiplicity 1. Consequently, they can be computed with higher accuracy. For this polynomial we are only interested in the roots lying on the unit circle where we again choose a separating annulus. If the number of roots of $H(x)$ on the unit circle is twice the number of those of $H'(x)$ these can be immediately used as an improved estimate. From these accurate roots we compute the polynomial coefficients of $P(x)$. It cannot be overemphasized that this has to be done with care. This apparently simple procedure may lead to completely perturbed polynomial coefficients even in well conditioned cases. The key idea to avoid these errors is a special ordering scheme known as bit reversal or — more sophisticated — Leja ordering [6].

We have implemented this approach in a MATLAB file and give an example by using the polynomial $P(x) = P_1(x)$ with degree $n = 100$. We computed $H(x)$ according to Eq. (5) and the corresponding minimum phase part according to the method above. The maximum error of the roots of the resulting polynomial compared to the original one is $1.4 \cdot 10^{-14}$, which is hardly worse than the achieved accuracy of the original polynomial ($5 \cdot 10^{-15}$). This is a considerable improvement since the roots of $H(x)$ on the unit circle can be determined only with accuracy $6 \cdot 10^{-9}$ with a straightforward use of our new root finder.

5 Conclusion

In addition to the examples given in Section 3 we evaluated the three programs by using all the other test polynomials given in [5]. These include polynomials with several multiple roots, roots that are close to each other, and roots with very large and/or very small absolute values. The following conclusions hold for all the polynomials just mentioned [6].

All three methods yield comparable results regarding speed and accuracy when working with "difficult" low degree ($n < 20$) polynomials. For these the CPU time is on a low and comparable level. The accuracy of our method is always better than that of the eigenvalue method and better than that of the Jenkins/Traub method in most cases. For larger degrees ($n > 30 \ldots 40$) the accuracy of the Jenkins/Traub method drastically decreases and it yields useless results for $n > 60 \ldots 70$. The accuracy of our method is better than that of the eigenvalue method in every case sometimes by more than a factor of 1000 (cf. $P_2(x)$). Furthermore the CPU time of our method increases much more slowly than that of the eigenvalue method, e.g., it is faster than a factor of hundred for $n = 500$. This fact and the linearly increasing memory compared to a quadratic increase needed for the eigenvalue method makes it possible to find roots in reasonable time even for high degree polynomials ($\approx$ 1000s for a degree 10000 polynomial). A C version of the program is available by anonymous ftp from cml.rice.edu in pub/markus/software.

Furthermore we gave a powerful approach for spectral factorization which improves the accuracy considerably. We have briefly considered the inverse problem to factorization, i.e., finding the polynomial coefficients from the roots where large errors can result. We gave a simple method to minimize them nearly up to computer accuracy.

References

[1] Aliphas, Amnon, S. Shankar Narayan, and Allen M. Peterson. Finding the zeros of linear phase FIR frequency sampling filters. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31:729–734, June 1983.

[2] Chen Xiangkun, and Thomas W. Parks. Design of optimal minimum phase FIR filters by direct factorization. *EURASIP Signal Processing*, 10:369–383, 1986.

[3] Göttlicher, W. *Beiträge zur rechnergestützten Synthese verlustloser Zweitore*. PhD thesis, University of Erlangen-Nürnberg, Germany, 1983.

[4] Jenkins, M. A. Algorithm 493 zeros of a real polynomial. *ACM Transactions on Mathematical Software*, 1:178–, June 1975.

[5] Jenkins, M. A. and J. F. Traub. Principles of testing polynomial zerofinding programs. *ACM Transactions on Mathematical Software*, 1:26–34, March 1975.

[6] Lang, Markus. A new and efficient program for finding all polynomial roots. Technical Report 9308, Department of Electrical and Computer Engineering, Rice University, Houston, TX, 1993.

[7] Press, William H. et al. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 1992.

[8] Schmidt, C. E. and L. R. Rabiner. A study of techniques for finding the zeros of linear phase FIR digital filters. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25:96–98, February 1977.

[9] Smith, B. T. et al. *Matrix Eigensystem Routines — EISPACK Guide*, volume 6 of *Lecture Notes in Computer Science*. Springer, 1976.

[10] Steiglitz, Kenneth, and Bradley Dickinson. Phase unwrapping by factorization. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30:984–991, December 1982.

[11] Williamson, J. Transfer matrix factorization using product-form polynomial scattering parameters. Master's thesis, University of Manitoba, Canada, 1987.

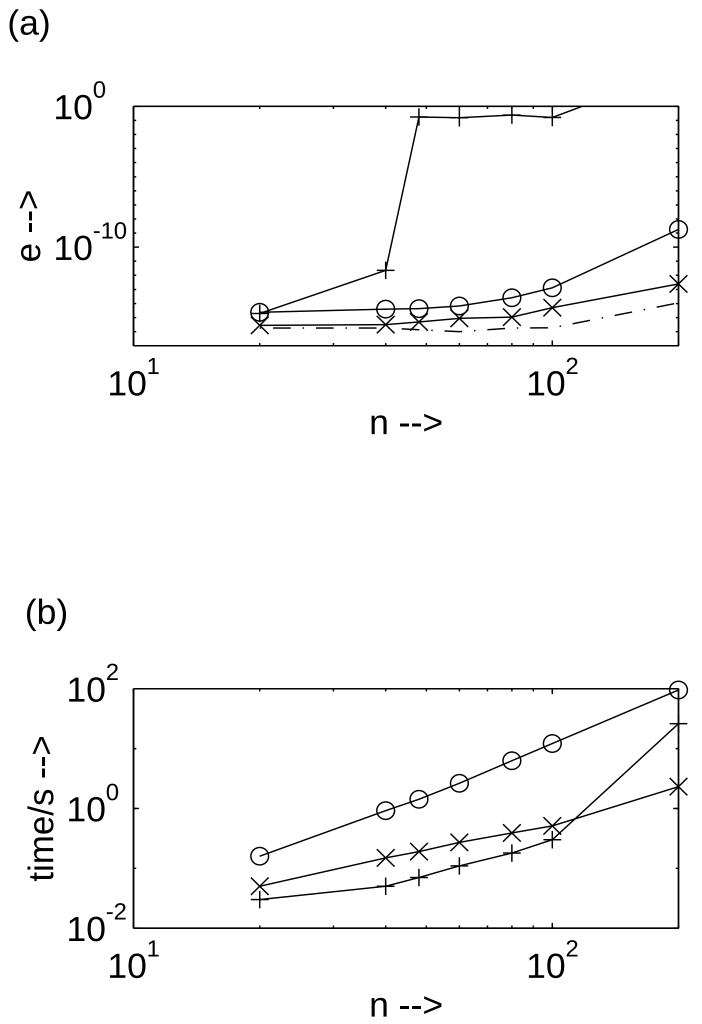

Figure 1:

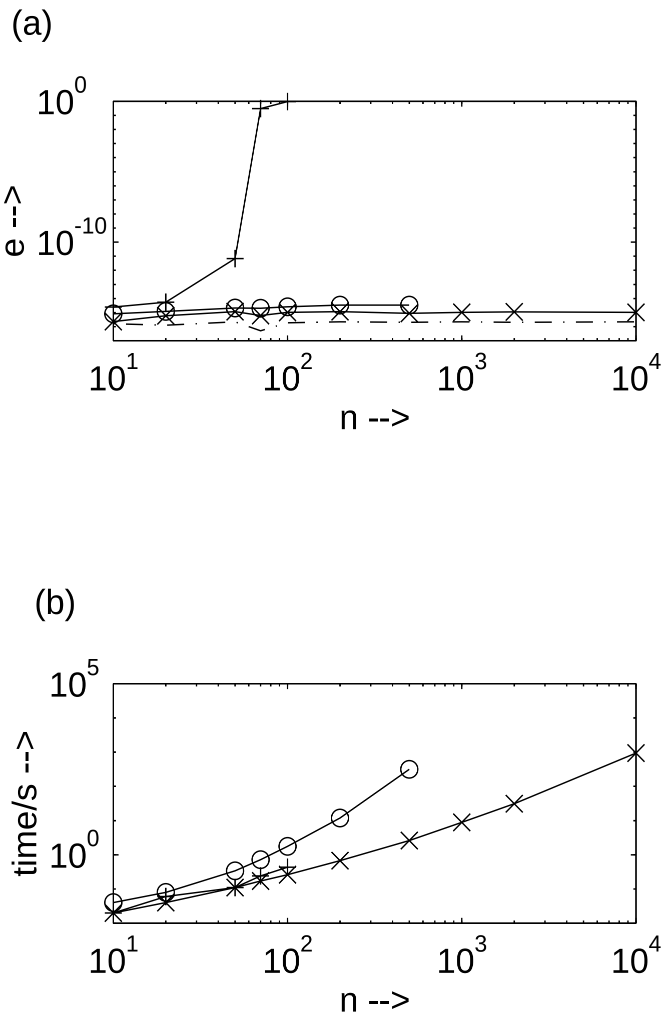

Figure 2: