Proc. of ICASSP-2008, pp. 3581-3584, Las Vegas, USA.

LEVINSON ALGORITHM OVER INTEGERS FOR STRONGLY REGULAR HERMITIAN TOEPLITZ MATRICES

Yaron Segalov and Yuval Bistritz

Department of Electrical Engineering
Tel Aviv University
Tel Aviv 69978, Israel
bistritz@eng.tau.ac.il

ABSTRACT

This paper presents a new version for the classical Levinson algorithm for solution of a symmetric (Hermitian) Toeplitz set of equations. The new version has the property that for a Toeplitz matrix with (Gaussian) integer entries the algorithm is carried out entirely over integers. The new algorithm has a low binary complexity with a near-linear integer growth rate. The integer preserving property provides an immediate means to control the numerical accuracy of the solution and its associated triangular factorization. It is also more attractive for symbolic computation.

Index Terms— Toeplitz matrices, Levinson algorithm, Linear prediction, Integer algorithms.

1. OVERVIEW

The Levinson Algorithm, originated in [1], is a fast method to solve a set of equations

$$\mathbf{R}_n[\alpha_0,\ldots,\alpha_{n-1},1]^T=[0,\ldots,0,E_n]^T \tag{1}$$

for the unknowns $\alpha_0\ldots\alpha_{n-1}$ and E_n, where $\mathbf{R}_n$ is a Hermitian Toeplitz matrix of the form

$$\mathbf{R}_n=\begin{pmatrix} r_0 & r_1 & \cdots & r_n \\ r_1^\star & r_0 & \cdots & r_{n-1} \\ \vdots & & \ddots & \\ r_n^\star & r_{n-1}^\star & \cdots & r_0 \end{pmatrix} \tag{2}$$

with $r_i\in\mathbb{C}$ (the field of complex numbers) where * denotes complex conjugate. This paper will assume that $\mathbf{R}_n$ is strongly regular, namely that the given set of equations as well as its subsets

$$\mathbf{R}_m[a_{m,0},\ldots,a_{m,m-1},1]^T=[0,\ldots,0,E_m]^T\ ,m=0,\ldots n \tag{3}$$

are nonsingular. The classical solution to the above set of equations for a strongly regular Hermitian $\mathbf{R}$ will be detailed as Algorithm 1 below.

The Levinson algorithm is a celebrated algorithm in signal processing and other algebraic problems. In signal processing, it is best known in the context of linear prediction. The setting in this case is $\mathbf{R}_n>0$ (positive definite) with $r_0=1,\ldots r_n\in\mathbb{R}$ (the real numbers) presenting the autocorrelation of a (stationary) process (or its estimates from measurements) whose current value $y[n]$ is best estimated from past values by the linear combination $\hat{y}[n]=-\sum_{i=0}^{n-1}\alpha_i y[n-i]$, and E_n is the minimal error autocorrelation (or the squared prediction error).

The classical algorithm is an efficient algorithm that exploits the Toeplitz structure to solve the above set of equations in $O(n^2)$ arithmetic operations. A more recent formulation called the *immittance* domain or *split* algorithm, takes a different recursion form and exploits some inherent symmetry in the problem to reduce the complexity by roughly a factor of two [2], [3]. For distinction of the original algorithm from the newer immittance, the classical algorithm is also called the *scattering* domain form. The two types of algorithms and their signal processing context are covered excellently in the two textbooks [4] and [5]. This paper will focus on the scattering domain Levinson algorithm.

The classical Levinson recursion obtains the solution to (1) for a strongly regular Hermitian Toeplitz matrix by solving successively the set of equations (3). We bring it here in polynomial notation using the polynomial sequence $a_m(z)=\sum_{i=0}^m a_{m,i}z^i,\quad m=0,\ldots,n$. We associate to a polynomial $a_m(z)$ the reciprocal polynomial denoted and defined by $a_m^\#(z)=\sum_{i=0}^m a_{m,m-i}^\star z^i$.

Algorithm 1 [The original Levinson algorithm].
Initiation. $a_0(z)=1$, $E_0=r_0$ and $\Delta_0=r_1$.
Recursion. For $m=1,\ldots,n$

$$k_m=\frac{\Delta_{m-1}}{E_{m-1}} \tag{4a}$$

$$a_m(z)=za_{m-1}(z)-k_m a_{m-1}^\#(z) \tag{4b}$$

$$E_m=E_{m-1}\left(1-|k_m|^2\right) \tag{4c}$$

$$\Delta_m=\sum_{i=0}^m a_{m,i}r_{i+1} \tag{4d}$$

Termination. The solution to (1) consist of $\{\alpha_0, \ldots, \alpha_{n-1}, 1\} = \{a_{n,0}, \ldots, a_{n,n-1}, 1\}$ and E_n.

Algorithm 1 is an efficient algorithm that may be run on a computer but it is not designed for any specific arithmetic environment. This paper will derive a new version for this algorithm that is designed to perform optimally in integer arithmetical environment. Recent years attest a growing interest in reexamination of various algebraic algorithms from a computer algebraic aspect. This led to adaptation of several classical algorithms to perform over integers beginning in [6] with Euclid's algorithm. Close to the current context of the Levinson algorithm are the version of the Jury stability test in[7], the Bistritz stability test in [8] and solving a Hankel set of equations [9]. An integer algorithm is not limited to but excels in integer environment or in the presence of symbolic parameters with symbolic computation packages. In our simple illustrative example for our new Levinson algorithm, we shall illustrate how it can be readily applied to combat degradation in numerical accuracy in floating point calculation due to rounding error.

It is also well known that Algorithm 1 also produces a triangular factorization of the inverse of the Toeplitz matrix. We can write (following the setting in [10]) $\mathbf{R}_n$,

$$\mathbf{R}_n^{-1} = \mathbf{A}\mathbf{\Lambda}^{-1}\mathbf{A}^H \tag{5a}$$

where the columns of $\mathbf{A}$ are the sequence of solution vectors for (3)

$$\mathbf{A} = \begin{pmatrix} 1 & a_{1,0} & \cdots & a_{n,0} \\ 0 & 1 & \cdots & a_{n,1} \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{5b}$$

and $\mathbf{\Lambda}$ is a diagonal matrix with the corresponding sequence of E_m's

$$\mathbf{\Lambda} = \begin{pmatrix} E_0 & 0 & \cdots & 0 \\ 0 & E_1 & \cdots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & E_n \end{pmatrix} \tag{5c}$$

Naturally, our integer version for the Levinson algorithm will also imply a corresponding triangular factorization over the integers for the inverse of an integer Toeplitz matrix.

2. THE INTEGER LEVINSON ALGORITHM

Suppose that we are given the set of equations (1) for a matrix $\mathbf{R}_n$ with integer entries. That is, $r_i \in \mathbb{Z}$ (the integers) if $\mathbf{R}_n$ is real or $r_i \in \mathbb{Z}_{[\mathbf{i}]}$ (the Gaussian integers) if $\mathbf{R}_n$ is complex. We are looking for an efficient way to solve such a set of equations over integers. That is, we wish to complete all computations (as far as possible) without resorting to the quotient field of rational numbers. We shall call an algorithm with this property integer-preserving (IP). To start, we rewrite (1) as

$$\mathbf{R}_n[\phi_0, \ldots, \phi_n]^T = [0, \ldots, 0, \tilde{E}]^T \tag{6}$$

and we seek a solution such that $\tilde{E}, \phi_i \in \mathbb{Z}$ for real $\mathbf{R}$ and $\phi_m \in \mathbb{Z}_{[\mathbf{i}]}$ when $\mathbf{R}$ is complex. As we freed the solution vector from its previous fixation to monic polynomial (ϕ_n is no longer fixed), the unknowns are unique only up to a common (integer) scaling factor. Among all possible solutions, the naturally desirable solution is the one with "smallest" integers. Concurrently, we want to find this small sized integer solution by an efficient algorithm that performs in a systematic manner that is not dependent on the input numbers. This at once rules out the idea of first finding somehow *any* integer solution, and then obtain the minimal sized solution by removing common integers using the oldest of all algorithm - Euclid's greatest common divisor algorithm (gcd) for integers. The worst part of first reaching a solution with unnecessarily large integers is an enormous increase of the computational cost. Our intermediate division-free algorithm will illustrate this point. The use of gcd, step by step, would both increase computation as well as produce a minimal sized integer solution that depends also on specifics of the input matrix. Instead, this paper will culminate on a revised form of the classical Levinson algorithm that works over integers and has an inherently restricted increase of coefficients size from step to step. Consequently, it produces systematically a small integers solution to (6).

For the following, we need to set (at least qualitatively) some measure for the size of an integer. A qualifying measure for the length $\ell(\phi)$ of an integer $\phi \in \mathbb{Z}_{[\mathbf{i}]}$ may be the number of bits that is required for binary presentation of $|\phi|$. It may also be the number of its decimals or any other measure that behaves similar to $\log|\phi|$ for addition and multiplication of integers. Correspondingly, for an integer polynomial $\phi(z) = \sum_{i=0}^{n} \phi_i z^i$, we shall use $\ell(\phi(z)) = max_i \ell(\phi_i)$ to measure the size of its coefficients.

The simplest way to turn Algorithm 1 into an algorithm over integers is by denying all divisions. We shall call this extension the *Division-Free* version and denote the sequence of polynomials it produces by $\{d_m(z) = \sum_{i=0}^{m} = d_{m,i} z^i, m = 0, \ldots, n\}$.

Algorithm 2 [Division-Free Levinson Algorithm].
Initiation. $d_0(z) = 1$, $\check{E}_0 = r_0$ and $\check{\Delta}_0 = r_1$.
Recursion. For $m = 1, \ldots, n$

$$d_m(z) = z\check{E}_{m-1} d_{m-1}(z) - \check{\Delta}_{m-1} d_{m-1}^{\#}(z) \tag{7a}$$

$$\check{E}_m = \check{E}_{m-1}^2 - |\check{\Delta}_{m-1}|^2 \tag{7b}$$

$$\check{\Delta}_m = \sum_{i=0}^{m} d_{m,i} r_{i+1} \tag{7c}$$

Termination. $\{\alpha_0, \ldots, \alpha_{n-1}, 1\} = \{d_{n,0}, \ldots, d_{n,n}\}/d_{n,n}$ and $\check{E}_n/d_{n,n}$.

It is evident that Algorithm 2 is integer preserving. If all $r_i \in \mathbb{Z}_{[\mathbf{i}]}$ then all $d_{m,i} \in \mathbb{Z}_{[\mathbf{i}]}$ simply because the algorithm

involves no divisions. Algorithm 2 however has an exponential coefficient growth, as it follows readily from (7a) that $\ell(d_m(z)) > 2\ell(d_{m-1}(z))$. Next, we show that common factors exist in all coefficients of $d_m(z)$. Then we shall see how they can be eliminated recursively and lead to an IP algorithm with more restrained growth of coefficients.

Proposition 1 (Common factors in the division-free recursion)**.** *With the onset of algorithm 2, $\check{E}_m$ divides all polynomials from $d_{m+2}(z)$ and onward.*

$$\check{E}_m \;\mid\; d_{m+2+j}(z) \quad \forall j \geq 0 \tag{8}$$

Proof. We perform two division-free steps (7) with equalities *modulo* $\check{E}_m$. We shall denote $(mod\ \check{E}_m)$ equality by $\cong$ for convenience.

$$\begin{aligned} d_{m+1,i} &= \check{E}_m d_{m,i-1} - \check{\Delta}_m d^\star_{m,m-i} \cong -\check{\Delta}_m d^\star_{m,m-i} \\ \check{E}_{m+1} &= \check{E}_m^2 - |\check{\Delta}_m|^2 \cong -|\check{\Delta}_m|^2 \\ \check{\Delta}_{m+1} &= \sum_{j=0}^{m+1} d_{m+1,j} r_{j+1} \cong -\check{\Delta}_m \sum_{j=0}^{m+1} d^\star_{m,m-j} r_{j+1} \end{aligned}$$

Another step of (7a) results in

$$d_{m+2,i} = \check{E}_{m+1} d_{m+1,i-1} - \check{\Delta}_{m+1} d^\star_{m+1,m+1-i}$$

$$\cong \check{\Delta}_m^2 \check{\Delta}^\star_m d^\star_{m,m+1-i} - \check{\Delta}_m \check{\Delta}^\star_m d_{m,i-1} \sum_{j=0}^{m} d^\star_{m,m-j} r_{j+1}$$

$$\cong |\check{\Delta}_m|^2 \sum_{j=0}^{m} (d^\star_{m,m+1-i} d_{m,j} - d_{m,i-1} d^\star_{m,m-j}) r_{j+1}$$

Analyzing the expression within brackets inside the summation, using (7a) *backwards* for all four elements, and then some algebra we can show that

$$\begin{aligned} &d_{n,i} d^\star_{n,j} - d^\star_{n,n-i} d_{n,n-j} = \\ &\left(\check{E}^2_{n-1} - |\check{\Delta}_{n-1}|^2\right) \left(d_{n-1,i-1} d^\star_{n-1,j-1} - d^\star_{n-1,n-1-i} d_{n-1,n-1-j}\right) \end{aligned}$$

Hence we conclude from (7b) and the previous equations that $d_{m+2,i} \cong 0$. From this point on, since $\check{E}_{m+2} = \check{\Delta}_{m+2} \equiv 0(mod\ \check{E}_m)$, clearly all subsequent polynomials vanish. □

Using this last result, the following improved IP variant of the Levinson algorithm can be stated. The sequence of polynomials produced by this algorithm will be denoted by $\{f_m(z) = \sum_{i=0}^{m} = f_{m,i} z^i, m = 0, \ldots, n\}$.

Algorithm 3 [Fraction-Free Levinson Algorithm].
Initiation. $f_0(z) = 1$, $\epsilon_0 = r_0$, $\delta_0 = r_1$ and $\epsilon_{-1} = 1$.
Recursion. For $m = 1, \ldots, n$ do:

$$f_m(z) = \frac{1}{f_{m-1,m-1}} \Big[\epsilon_{m-1} z f_{m-1}(z) - \delta_{m-1} f^{\#}_{m-1}(z) \Big] \tag{9a}$$

$$\epsilon_m = \frac{\epsilon^2_{m-1} - |\delta_{m-1}|^2}{\epsilon_{m-2}} \tag{9b}$$

$$\delta_m = \sum_{i=0}^{m} f_{m,i} r_{i+1} \tag{9c}$$

Termination. The solution to (1) is given by $\alpha_{n,i} = f_{n,i}/f_{n,n}$ $i = 0, \ldots, n$, and $E_n = \epsilon_n / f_{n,n}$.

Note that inspecting the leading coefficient in (9a) reveals that $f_{m,m} = \epsilon_{m-1}$, so that the two can be used interchangeably.

Proposition 2. *Algorithm 3 is integer preserving. Namely, if $r_m \in \mathbb{Z}_{[\mathbf{i}]}, m = 0, \ldots, n$ then all $f_{m,i} \in \mathbb{Z}_{[\mathbf{i}]}$ and all $\epsilon_m \in \mathbb{Z}$.*

Proof. Algorithm 3 follows from the integer-preserving Algorithm 2 after carefully noting that the common factors exposed in Proposition 1 can also be removed recursively. This implies $f_{m,i} \in \mathbb{Z}_{[\mathbf{i}]}$. To realize that $\epsilon_m \in \mathbb{Z}$ observe that since r_0 is real (by the assumption that $\mathbf{R}$ is Hermitian) all $d_{m,m}$, $\check{E}_m$, $f_{m,m}$ are real also in the complex case. □

Proposition 3. *The coefficients of $f_m(z)$ produced by Algorithm 3 are upped-bounded by $\ell(f_m(z)) = mB + \frac{1}{2} m \log(m)$, $m = 1, \ldots, n$, where B is the bound for the largest entry of the integer matrix.*
The proof is beyond the current scope.

Triangular factorization. Next, we wish to use the results of algorithm 3 for reaching an IP version of the factorization (5) using the following triangular and diagonal matrices:

$$\begin{aligned} \mathbf{F} &= \begin{pmatrix} f_{0,0} & f_{1,0} & \cdots & f_{n,0} \\ 0 & f_{1,1} & \cdots & f_{n,1} \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & f_{n,n} \end{pmatrix} \\ \mathbf{E} &= diag\left[\ \epsilon_0, \quad \epsilon_1, \quad \ldots \quad , \epsilon_n\ \right] \end{aligned}$$

The scale of the fraction-free polynomial $f_m(z)$ with regard to the monic polynomial $a_m(z)$ at each recursion step is $f_{m,m}$ known to be equal to ϵ_{m-1}. Thus

$$\begin{aligned} \mathbf{F} &= \mathbf{GA} \\ \mathbf{G} &= diag\left[\ 1, \quad \epsilon_0, \quad \cdots \quad , \epsilon_{n-1}\ \right] \end{aligned} \tag{10}$$

and applying to (5), the IP triangular factorization becomes

$$\begin{aligned} \mathbf{R}^{-1} &= \mathbf{F}(\mathbf{GE})^{-1}\mathbf{F}^H = \mathbf{F}\mathbf{D}^{-1}\mathbf{F}^H \\ \mathbf{D} = \mathbf{GE} &= diag\left[\ \epsilon_0, \quad \epsilon_1\epsilon_0, \quad \cdots \quad , \epsilon_n\epsilon_{n-1}\ \right] \end{aligned} \tag{11}$$

Numerical Illustration. Rather then just bringing a numerical example that describes the new algorithm for the case of an integer Toeplitz matrix, let us also illustrate how it can readily be used to increase numerical accuracy. Suppose we are given a set of equations (1) with decimal numbers,

$$\begin{pmatrix} 1 & 0.8 & 0.6 & 0.4 & 0.2 \\ 0.8 & 1 & 0.8 & 0.6 & 0.4 \\ 0.6 & 0.8 & 1 & 0.8 & 0.6 \\ 0.4 & 0.6 & 0.8 & 1 & 0.8 \\ 0.2 & 0.4 & 0.6 & 0.8 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ Q \end{pmatrix} \tag{12}$$

We can scale it into an integer matrix and then solve it by the IP algorithm. In any computational environment that accepts instruction for integer format this approach may be enough to proceed to a solution without further degradation in numerical accuracy caused by rounding errors. The given Toeplitz matrix with decimal entries, say $\mathbf{T}$, can be converted into an integer matrix $\mathbf{R} = fac \cdot \mathbf{T}$ by some scaling factor. In the above example, taking $fac = 5$, gives $\mathbf{R}_4 = 5\mathbf{T}_4$,

$$\mathbf{R}_4 = \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 5 & 4 & 3 & 2 \\ 3 & 4 & 5 & 4 & 3 \\ 2 & 3 & 4 & 5 & 4 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

The application of Algorithm 3 to solve (1) with the above $\mathbf{R}_4$ begins with $f_0(z) = 1$; $\epsilon_0 = 5$; $\delta_0 = 4$. The recursion results in

$$\begin{array}{lll} f_1(z) = 5z - 4; & \epsilon_1 = 9; & \delta_1 = -1 \\ f_2(z) = 9z^2 - 8z + 1; & \epsilon_2 = 16; & \delta_2 = -2 \\ f_3(z) = 16z^3 - 14z^2 + 2; & \epsilon_3 = 28; & \delta_3 = -4 \\ f_4(z) = 28z^4 - 24z^3 + 4; & \epsilon_4 = 48 & \end{array}$$

Thus, the solution to (1) or (12) is given by the coefficient vector of the polynomial $\alpha(z) = f_4(z)/28 = \frac{1}{28}[4\ ,\ 0\ ,\ 0\ ,\ -24\ ,\ 28]^T$, and $E_4 = \epsilon_4/28 = 12/7$ or $Q = E_4/fac = 12/35$, respectively. The algorithm also provides the triangular factorization (11) for $\mathbf{R}_4^{-1}$ with

$$\mathbf{F} = \begin{pmatrix} 1 & -4 & 1 & 2 & 4 \\ 0 & 5 & -8 & 0 & 0 \\ 0 & 0 & 9 & -14 & 0 \\ 0 & 0 & 0 & 16 & -24 \\ 0 & 0 & 0 & 0 & 28 \end{pmatrix}$$

$$\mathbf{D} = \begin{pmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 45 & 0 & 0 & 0 \\ 0 & 0 & 144 & 0 & 0 \\ 0 & 0 & 0 & 448 & 0 \\ 0 & 0 & 0 & 0 & 1344 \end{pmatrix}$$

Since $\mathbf{T}^{-1} = fac \cdot \mathbf{R}^{-1}$, the integer triangular factorization of $\mathbf{R}^{-1}$ can be used also to write a factorization for the inverse of the decimal matrix.

Running the classical Algorithm 1 for $\mathbf{T}_4$ in floating numbers (on Matlab ®) gives, already for this low order system, a less accurate solution with the following numbers for the factorization (5) of $\mathbf{T}^{-1}$

$$\mathbf{\Lambda} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 3.6000e-001 & 0 & 0 & 0 \\ 0 & 0 & 3.5556e-001 & 0 & 0 \\ 0 & 0 & 0 & 3.5000e-001 & 0 \\ 0 & 0 & 0 & 0 & 3.4286e-001 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 1 & -8.0000e-001 & 1.1111e-001 & 1.2500e-001 & 1.4286e-001 \\ 0 & 1 & -8.8889e-001 & 7.4940e-016 & -4.4409e-016 \\ 0 & 0 & 1 & -8.7500e-001 & 8.5646e-016 \\ 0 & 0 & 0 & 1 & -8.5714e-001 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Notice, how the solution to (12), given by the last column of $\mathbf{A}$ with Q given by the last entry of the diagonal matrix $\mathbf{\Lambda}$, differs from its solution by the new algorithm (e.g. how it misses $a_{4,1} = a_{4,2} = 0$).

We remind that the new algorithm was derived for a Toeplitz matrix with complex entries. Space limitation does not admit a second numerical illustration with Gaussian ('complex') integers.

3. SUMMARY

An efficient integer-preserving version of the classical Levinson algorithm was presented. It features a restrained (almost linear) growth rate for the size of integers with an implied high binary efficiency. A computer implementation of this algorithm in an integer environment can benefit from smaller computational load, due to the restricted increase of coefficient size (whose maximum size may be pre-determined from the inputs). The new version is more suitable than the classical algorithm to handle computation with symbolic parameters and is immune to rounding error that in the classical algorithm is noticeable already with small input sizes.

4. REFERENCES

[1] N. Levinson, "The Wiener RMS error criterion in filter design and prediction", *J. Math. Phys.*, vol. 25, pp. 261-278, 1947.

[2] Y. Bistritz, H. Lev-Ari, and T. Kailath "Immittance domain Levinson algorithms," *IEEE Trans. Information Theory,* vol. 35 (3), pp. 674-682, May 1989.

[3] P. Delsarte and Y. Genin, "The split levinson algorithm", *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. ASSP-34 (3), pp. 470-478, June 1986.

[4] S. J. Orfanidid *Optimum signal processing: An introduction*, 2nd Edition, MacMillan, 1988.

[5] C. W. Therrien, *Discrete random signals and statistical signal processing*, Prentice-Hall, 1992.

[6] W. S. Brown and J. F. Traub, "On Euclid's algorithm and the theory of sub-resultants", *J. ACM*, vol. 18, pp. 505-514, 1971.

[7] P. G. Anderson, M.R. Garey and L.E. Heindel, "Computational aspects of deciding if all roots of a polynomial lie within the unit circle", *Computing* vol. 16, pp. 293-304, 1976.

[8] Y. Bistritz, "An efficient integer-preserving stability test for discrete-time systems", *Circuits Systems Signal Processing*, vol. 23 (3), pp. 195-213, 2004.

[9] L. Gemignani, "Solving Hankel systems over the integers", *J. Symbolic Computation*, vol. 18, pp. 573-584, 1994.

[10] Y. Bistritz, "Reflection on Schur-Cohn matrices and Jury-Marden tables and classification of related unit circle zero location criteria" *Circuits Systems Signal Processing*, vol. 15, (1), pp. 111-136, 1996.