

Algorithms for speech coding systems based on linear prediction

Citation for published version (APA):

Rooijackers, J. E. (1992). *Algorithms for speech coding systems based on linear prediction*. (EUT report. E, Fac. of Electrical Engineering; Vol. 92-E-260). Eindhoven University of Technology.

Document status and date:

Published: 01/01/1992

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.



Research Report

ISSN 0167-9708

Coden: TEUEDE

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

Algorithms for Speech Coding Systems Based on Linear Prediction

by
J.E. Rooijackers

EUT Report 92-E-260
ISBN 90-6144-260-5
july 1992

Eindhoven Universit of Technology Research Reports

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
Eindhoven The Netherlands

ISSN 0167-9708

Coden:TEUEDE

ALGORITHMS FOR SPEECH CODING SYSTEMS
BASED ON LINEAR PREDICTION

by
J.E. Rooijackers

EUT Report 92-E-260
ISBN 90-6144-260-5

EINDHOVEN
July 1992

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Rooijackers, J.E.

Algorithms for speech coding systems based on linear
prediction / J.E. Rooijackers. - Eindhoven : Eindhoven
University of Technology, Faculty of Electrical
Engineering. - Fig. - (EUT report, ISSN 0167-9708 ;
92-E-260)

Met index, lit. opg.

ISBN 90-6144-260-5

NUGI 832

Trefw.: spraaksynthese (computertechniek).

Abstract

This report presents a set of algorithms to tailor speech coding systems, that are based on linear prediction. The mathematical background of the algorithms is treated and the source text of the algorithms is given. Special attention is given to the inter relations between the methods and to the computational efficiency.

Rooijackers, J.E.

ALGORITHMS FOR SPEECH CODING SYSTEMS BASED ON LINEAR PREDICTION

Faculty of Electrical Engineering, Eindhoven University of Technology,

The Netherlands, 1992.

EUT Report 92-E-260

Address of the author:

The Group of Information and Communication Theory,

Faculty of Electrical Engineering,

Eindhoven University of Technology,

P.O. Box 513

5600 MB EINDHOVEN,

The Netherlands.

Contents

Preface	vi
1 Introduction to prediction	1
1.1 Signal models	1
1.2 Signal processing	3
1.3 Speech coding and adaptivity	4
1.4 Spectrum analysis	5
2 Linear prediction	6
2.1 The Yule-Walker equation and the Levinson-Durbin recursion	6
2.2 The Levinson and related algorithm.	8
2.3 Analysis and synthesis filters	10
3 The Schur algorithms	14
4 The Line Spectrum Pairs (LSP)	17
5 Other optimization criteria	26
5.1 The autocorrelation method	27
5.2 The covariance method	29
5.3 The Burg algorithm	30
5.4 The Marple algorithm	33
5.5 The Morf algorithm	42
APPENDICES	
A Mathematical preliminaries	47
A.1 Review of linear spaces and inner products	47
A.2 The projection theorem	49
A.3 Orthogonality principle revisited	49
B The Levinson-Durbin recursion in matrix form	51
B.1 The symmetric or Hermitian Toeplitz situation	51
B.2 The non-symmetric Toeplitz situation	52
B.3 The physical meaning of several quantities	55
C The Cholesky decomposition	57
D Procedures in TURBO PASCAL	60
Bibliography	83
Index	86

Preface

The emerging application of compressed speech in telecommunication services have renewed the interest for speech coding algorithms. Research in speech coding has been active for twentyfive years but the introduction of these techniques in operational systems has been slow and difficult. The main limitations were the hardware complexity connected with the implementation of the speech coders and the quality which was judged unsatisfactory by the service operators. Two factors that brought a breakthrough were the introduction on the component market of the digital signal processor (DSP) chips and the studies on the analysis-synthesis algorithms. Some of the resently adapted systems are a 32 kBit/s CCITT approved ADPCM system, a 16 kBit/s APC for Inmarsat standard B, 13 kBit/s LPC system for a pan-European digital mobile radio system selected by the CEPT, the 9.6 kBit/s system for airline passengers communication in the Avsat and Skyphone systems and a 4.8 kBit/s of the NASA advanced mobile vehicle-satellite radio channels.

The purpose of this report is multiple :

- An introduction to speech coding systems based on a signal model and as a consequence, based on linear prediction.
- A mathematical background for linear prediction, the most important model parameters and the algorithms to obtain these parameters in a computational efficient way.
- A description of the available algorithms and their inter relations.
- A presentation of a complete software package that covers all the algorithms and their combinations. This package is written in the language Turbo Pascal.

The object is not to describe several speech coding systems in detail, but to understand them and to recognize and comprehend their kinship. With the algorithms, given in this report, the existing coding systems can be upgraded or more enhanced systems can be developed.

The signal model used in the report is the autoregresive (AR) model. This reveals items as predictors, analysers and synthesizers, which can be described by parameters known as the reflection coefficients or the partial correlation (PARCOR) coefficients. If linear prediction is applied with the mean-square error (MSE) criterion the Levinson-Durbin recursion is the result and the Levinson algorithms are found. The analyse and synthesis filters can be realized as lattice filters, a more robust form compared with the transversal filters. This can be important for the VLSI realization. For parallel processing the Schur algorithms offer even more efficient computational possibilities. The parcor coefficients can be replaced by more powerfull parameters as the line spectrum pairs (LSP). If the MSE criterion is changed into the more practical least total square error (LSE) criterion, two methods are found. For the first method, the so called correlation method, the previous mentioned results are valid

because the autocorrelation matrix is Toeplitz. The second method, the covariance method, has the advantage that the data is not windowed. For this last method several algorithms are derived, such as the Cholesky, Burg, Morf and Marple algorithm.

The report¹ is organized as follows. In chapter 1 an introduction is given to linear prediction, data processing and speech coding systems. Chapter 2 treats the linear prediction in more detail. Emphasis is placed on subjects as the Yule-Walker equation, the Levinson-Durbin recursion, the Levinson algorithms and the realization of the analyse/synthesis filters. Chapter 3 introduces the Schur algorithms and chapter 4 gives an introduction of the line spectrum pairs and several algorithms to obtain these parameters are described. In chapter 5 the (auto)correlation and covariance methods for parameter estimation are given and algorithms, based on the LSE criterion, are treated. Three appendices (A, B and C) form a backup of the theory, while appendix D gives the source text, written in the Turbo Pascal language, of the algorithms.

¹The chapters 1 and 2 are presented at the First Benelux-Japan Workshop on Information and Communication Theorie, Eindhoven, The Netherlands, September 1989.

Chapter 1

Introduction to prediction

In this chapter a connection between signal modelling, linear prediction and spectrum estimation will be made. This gives a theoretical background for existing signal and speech processing methods. In the past these methods were invented in a more or less ad hoc way, but now a motivation can be given.

1.1 Signal models

One of the most useful ways to model a (random) signal is to consider it as being the output of a causal and stable filter $B(z)$ which is driven by a stationary uncorrelated (white-noise) sequence $\{\varepsilon_0, \varepsilon_1, \dots, \varepsilon_n, \dots\}$ with an autocorrelation function

$$R_{\varepsilon\varepsilon}(k) = \mathbb{E}[\varepsilon_n \varepsilon_{n+k}] = \sigma_\varepsilon^2 \delta(k). \quad (1.1)$$

The output signal y_n is obtained by convolving the input sequence ε_n with the filter's impulse

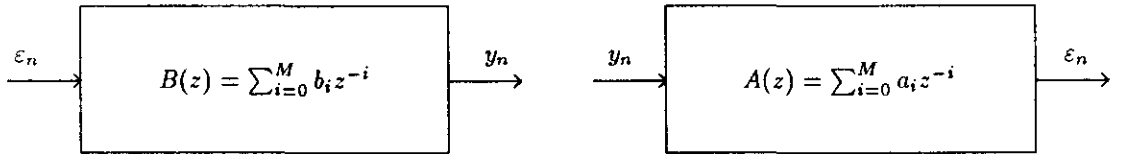


Figure 1.1: The synthesis filter and the analysis filter.

response b_n

$$y_n = \sum_{i=0}^M b_i \varepsilon_{n-i}, \quad n = 0, 1, 2, \dots \quad (1.2)$$

In these formulas M and σ_ε^2 are the order of the model and the variance of the noise respectively. The power spectrum of the output sequence is

$$S_{yy}(\omega) = \sigma_\varepsilon^2 |B(\omega)|^2. \quad (1.3)$$

The stability of the filter $B(z)$ is essential as it guarantees the stationarity of the sequence y_n . If we write the synthesis filter $B(z)$ as the ratio of two polynomials

$$B(z) = \frac{N(z)}{D(z)}, \quad (1.4)$$

then the stability and the causality restriction requires that the zeros of the polynomial $D(z)$ lie inside the unit circle in the complex z -plane. The filter of (1.4) is called an auto regressive moving average (ARMA) or a pole-zero model. Two special cases of interest are the moving average (MA) or all-zero model if $B(z) = N(z)$ and the auto regressive (AR) or all-pole model with $B(z) = \frac{1}{D(z)}$.

To synthesize a physical signal, for example speech, we need some analysis algorithm to determine the model parameters $\{b_1, b_2, \dots, b_M, \sigma_e^2\}$ and a method to obtain the excitation sequence ε_n . This excitation signal is generated by passing the (speech) signal through an inverse filter of the form

$$A(z) = \frac{1}{B(z)} \quad (1.5)$$

as is depicted in the righthand side of Figure 1.1. Note that the filter parameter b_0 is ignored, because by readjusting the value σ_e^2 we may assume $b_0 = 1$. For $A(z)$ to be stable and causal requires (see 1.4 and 1.5) the zeros of $N(z)$ to be inside the unit circle. Thus, both the poles and zeros of $B(z)$ must lie inside the unit circle. Such filters are called minimal phase filters. In the sequel of this report the AR-model will be treated, unless stated otherwise. Other names for the inverse filter are analysis filter, whitening filter or prediction-error filter.

The filters $A(z)$ and $1/A(z)$ can be realised with linear prediction of order M (Figure 1.2). Taking the z -transform of the sequences, y_n , \hat{y}_n and ε_n of the lefthand part of Figure 1.2 we

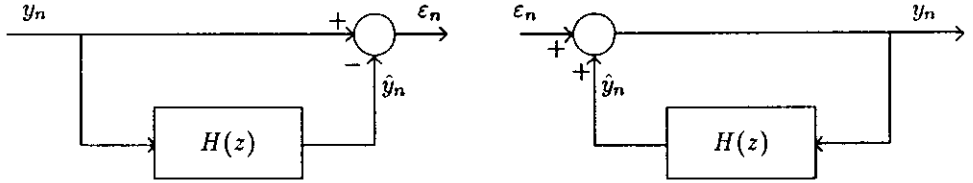


Figure 1.2: The forward predictor $A(z)$ and the backward predictor $1/A(z)$.

obtain

$$E(z) = Y(z) - \hat{Y}(z) = Y(z)\{1 - H(z)\} = A(z)Y(z), \quad (1.6)$$

with

$$H(z) = - \sum_{i=1}^M a_i z^{-i}, \quad (1.7)$$

and thus

$$A(z) = 1 - H(z) = \sum_{i=0}^M a_i z^{-i}. \quad (1.8)$$

So ε_n becomes

$$\varepsilon_n = \sum_{i=0}^M a_i y_{n-i}, \quad a_0 = 1, \quad (1.9)$$

which is the same expression as the one obtained for the output sequence of the analysis filter of Figure 1.1. For the signal \hat{y}_n we find

$$\hat{y}_n = - \sum_{i=1}^M a_i y_{n-i}, \quad (1.10)$$

which is the linear prediction for the signal y_n and which is a linear combination of the M previous samples of y_n . The signal ϵ_n is the prediction error and the aim is to find those prediction coefficients $\{a_1, \dots, a_M\}$ that minimizes this error. It is easy to see that the righthand part of Figure 1.2 gives a realization of the synthesis filter $B(z) = 1/A(z)$.

We note here an interesting connection between linear prediction concepts and signal modeling concepts; namely, that the optimal linear predictor determines the analysis filter $A(z)$ which, in turn, determines the generator model $B(z) = 1/A(z)$ of y_n . In other words, the solution of the linear prediction problem is also the solution of the modeling problem.

1.2 Signal processing

If we call the analyser of Figure 1.2 the encoder and the synthesizer of Figure 1.1 the decoder and if we want an exact reproduction of y_n at the decoder both the model parameters $\{a_1, a_2, \dots, a_M, \sigma_\epsilon^2\}$ and the entire sequence ϵ_n must be stored or transmitted. But in data compression schemes known as differential pulse codemodulation (DPCM) or residual encoding [9] [22] the filters $A(z) = \frac{1}{B(z)}$ or $H(z)$ are fixed and the residual sequence ϵ_n is stored or transmitted with reduced accuracy. Each value of ϵ_n is quantized to one of 2^b levels, where b is the number of binary digits used to represent each value of ϵ_n . A complete DPCM system is shown in Figure 1.3 where we use the symbol e_n of the prediction error in stead of ϵ_n . The conversion from \tilde{e}_n to code words and visa versa is omitted. The presence of the quantizer

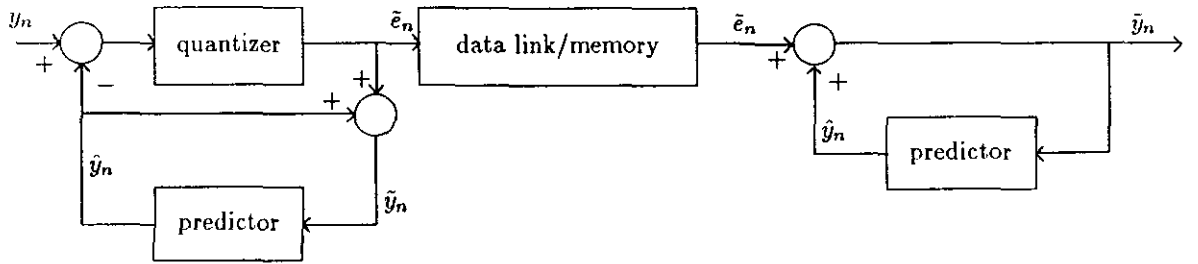


Figure 1.3: A DPCM-system.

introduces a quantizer error q_n such that

$$\tilde{e}_n = e_n + q_n. \quad (1.11)$$

The particular realization shown in Figure 1.3 ensures that, at the reconstruction end, the quantization errors do not accumulate because

$$\tilde{y}_n - y_n = (\tilde{e}_n + \hat{y}_n) - y_n = \tilde{e}_n - e_n = q_n. \quad (1.12)$$

The reconstruction error is equal to the quantization error.

To reduce the data link or memory capacity the residual signal can be omitted and replaced by a random number generator at the reconstruction or synthesizer side. A diagram of such a linear prediction coder (LPC) is shown in Figure 1.4. Here the quantized versions of the filter coefficients $\{a_1, a_2, \dots, a_M\}$ and of the variance or gain σ_e^2 are stored or transmitted. synthesis filters realised with quantized parameters are not guaranteed minimum phase. By

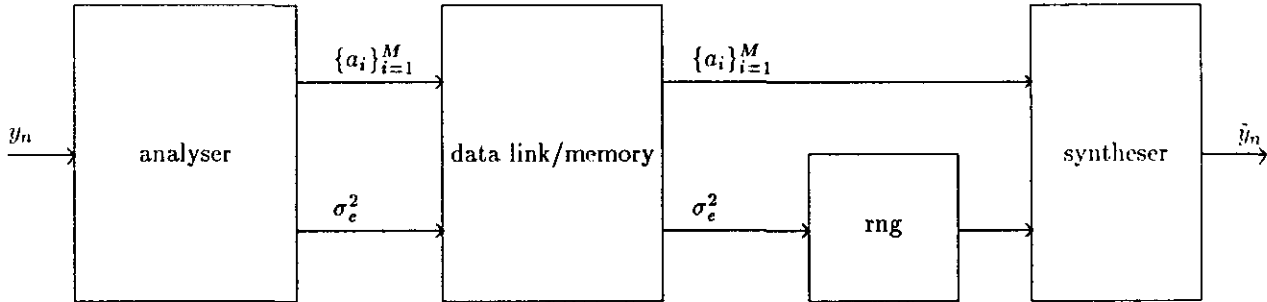


Figure 1.4: LPC or analyse/synthesis system.

using lattice filters in stead of transversal filters this problem is more easily attacked.

1.3 Speech coding and adaptivity

So far we have assumed stationarity of the signal y_n . But speech is a non-stationary signal, so some form of adaptivity is needed. In Figure 1.4 the analyse algorithm estimates the model parameters during a block of input samples. During this analyse frame the signal is assumed stationary. A more realistic representation of a speech frame requires the specification of two additional parameters : the pitch period and a voiced/unvoiced (V/UV) decision. Unvoiced sounds have a white-noise sounding nature and are generated by the turbulent flow of air through the constrictions of the vocal tract. Such sounds may be represented adequately by the random signal model. On the other hand, voiced sounds, such as vowels, are pitched sounds, and have a pitch period associated with them. They may be assumed to be generated by the periodic excitation of the vocal tract by a train of impulses separated by the pitch period. The vocal tract respond to each of these impulses by producing its impulse response, resulting therefor in a quasi-periodic output which is characteristic for such sounds. Thus, depending on the type of sound, the nature of the generator of the excitation input to the synthesis filter will be different. It will be a random number generator (rng) for unvoiced sounds and a pulse train for voiced sounds. A typical synthesis system is depicted in Figure 1.5. Using the block adaptive method with DPCM requires side information, so normaly a second method, recursive or sequential adaptivity, is used. The filter coefficients are continually adaptive to a signal both the encoder and the decoder have in common. This system, called adaptive differential pulse code modulation (ADPCM), has also an adaptive quantizer for the residual signal. Combinations of both systems consisting of block adaption of the filter parameters and coarse quantization of the residual signal are known as adaptive predictive coding (APC) [6].

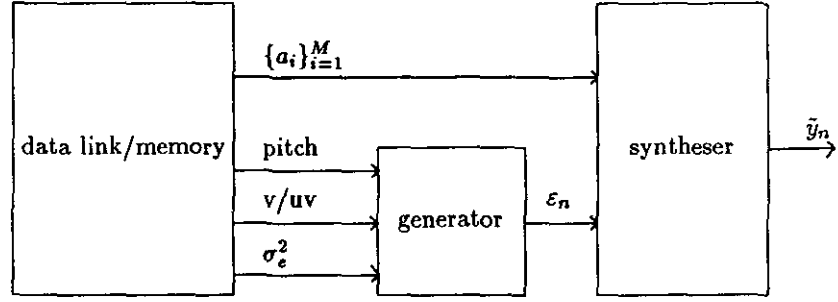


Figure 1.5: A speech synthesis system.

1.4 Spectrum analysis

As can be seen from (1.3) the spectral shape of the power spectrum of the sequence y_n , arises only from the spectral shape of the synthesis filter. For the AR model the problem of spectrum estimation can be linked to the problem of linear prediction. With (1.5) and (1.8) and for $z = e^{j\omega}$ the power spectrum of (1.3) becomes

$$S_{yy}(\omega) = \frac{\sigma_e^2}{|\sum_{i=0}^M a_i e^{-ji\omega}|^2}. \quad (1.13)$$

The specrum estimates based on such parametric models tend to have much better frequency resolution properties then the classical methods, especially when the length of the available data record is short [16] [19]. A classical approach is, for example, the direct computation of the Fourier transform of the data record.

Chapter 2

Linear prediction

In this chapter we will estimate the optimal p^{th} order linear predictor for a stationary signal with an autocorrelation function $R(k) = \mathbb{E}[y_n y_{n+k}]$. The prediction order p is an arbitrary number smaller than M . Iterative procedures will be found to determine the $(p+1)^{th}$ order predictor from the previous p^{th} order predictor. Also the lattice implementation of predictors will be introduced.

2.1 The Yule-Walker equation and the Levinson-Durbin recursion

The p prediction coefficients $a_{p,1}, a_{p,2}, \dots, a_{p,p}$ are chosen to minimize the mean-square prediction error

$$E_p = \mathbb{E}[e_p^2(n)], \quad (2.1)$$

where $e_p(n)$ is the prediction error (1.9)

$$e_p(n) = \sum_{i=0}^p a_{p,i} y_{n-i}, \quad a_{p,0} = 1. \quad (2.2)$$

Differentiating (2.1) with respect to each coefficient $a_{p,i}$, $i = 1, 2, \dots, p$, yields the orthogonality equations

$$\mathbb{E}[e_p(n) y_{n-i}] = 0, \quad i = 1, 2, \dots, p. \quad (2.3)$$

See also the projection theorem in Appendix A and section A.3.

Inserting (2.2) in (2.3) results in p linear equations

$$\sum_{j=0}^p a_{p,j} \mathbb{E}[y_{n-j} y_{n-i}] = \sum_{j=0}^p a_{p,j} R(|i-j|) = 0, \quad i = 1, 2, \dots, p \quad (2.4)$$

For the minimized value of (2.1) we find

$$E_p = \sigma_e^2 = \mathbb{E}[e_p^2(n)] = \mathbb{E}[e_p(n) y_n] = \sum_{i=0}^p a_{p,i} R(i). \quad (2.5)$$

Equations (2.4) and (2.5) can be combined into the $(p+1) \times (p+1)$ matrix equation

$$\begin{bmatrix} R(0) & R(1) & R(p) \\ R(1) & R(0) & R(p-1) \\ \vdots & \vdots & \vdots \\ R(p) & R(p-1) & R(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_{p,1} \\ \vdots \\ a_{p,p} \end{bmatrix} = \begin{bmatrix} \sigma_\epsilon^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (2.6)$$

Equation (2.6) is called the normal or Yule-Walker equation and can be solved directly by matrix inversion. Here we follow the Levinson-Durbin recursion method to obtain all the best linear predictions from $p = 1, p = 2$ to $p = M$ and to obtain the lattice realization of linear prediction filters. We see that the matrix of autocorrelation functions has identical elements along any diagonal and that the matrix is symmetric. It is called a symmetric Toeplitz matrix.

Suppose that the optimum predictor of order p with coefficients $1, a_{p,1}, \dots, a_{p,p}$ has already been constructed. The corresponding gapped function is

$$\begin{aligned} g_p(k) &= \mathbb{E}[e_p(n)y_{n-k}] = \mathbb{E}[(\sum_{i=0}^p a_{p,i}y_{n-i})y_{n-k}] \\ &= \sum_{i=0}^p a_{p,i}R(k-i). \end{aligned} \quad (2.7)$$

This function has a gap of length p , that is

$$g_p(k) = 0, \quad 1 \leq k \leq p. \quad (2.8)$$

It is easy to see that $g_p(p+1-k)$ has the same gap, and that a linear combination of both functions has a gap of p . Therefore

$$g_{p+1}(k) = g_p(k) - \gamma_{p+1}g_p(p+1-k) \quad (2.9)$$

has a gap of $p+1$ if we choose γ_{p+1} such that

$$g_{p+1}(p+1) = g_p(p+1) - \gamma_{p+1}g_p(0) = 0 \quad (2.10)$$

or

$$\gamma_{p+1} = \frac{g_p(p+1)}{g_p(0)} = \frac{\epsilon_p}{E_p} \quad (2.11)$$

where

$$E_p = g_p(0) = \mathbb{E}[e_p(n)y_n] = \mathbb{E}[e_p^2(n)] \quad (2.12)$$

and

$$\epsilon_p = g_p(p+1) = \sum_{i=0}^p a_{p,i}R(p+1-i). \quad (2.13)$$

Using (2.9) and (2.11) we find a recursion for the minimal mean-squared prediction error

$$E_{p+1} = g_{p+1}(0) = g_p(0) - \gamma_{p+1}(p+1) = (1 - \gamma_{p+1}^2)g_p(0) \quad (2.14)$$

or

$$E_{p+1} = (1 - \gamma_{p+1}^2)E_p. \quad (2.15)$$

Since both E_{p+1} and E_p are nonnegative it follows that

$$|\gamma_{p+1}| \leq 1. \quad (2.16)$$

The coefficient γ_{p+1} is called reflection, PARCOR or Schur coefficient. To obtain the prediction coefficients we take the z -transform of (2.7) for p and $p + 1$ and substitute the result in the z -transform of (2.9):

$$A_{p+1}(z)S_{yy}(z) = A_p(z)S_{yy}(z) - \gamma_{p+1}z^{-(p+1)}A_p(z^{-1})S_{yy}(z^{-1}). \quad (2.17)$$

Using the symmetry relation of $S_{yy}(z)$ we get the Levinson-Durbin recursion

$$A_{p+1}(z) = A_p(z) - \gamma_{p+1}z^{-(p+1)}A_p(z^{-1}). \quad (2.18)$$

Taking the z -transform of (2.18) gives

$$\begin{aligned} a_{p+1,i} &= a_{p,i} - \gamma_{p+1}a_{p,p+1-i}, \quad 1 \leq i \leq p, \\ a_{p+1,p+1} &= -\gamma_{p+1}. \end{aligned} \quad (2.19)$$

Introducing the reverse polynomial $A_p^r(z) = z^{-p}A_p(z^{-1})$ we may write (2.18) as

$$A_{p+1}(z) = A_p(z) - \gamma_{p+1}z^{-1}A_p^r(z). \quad (2.20)$$

From $A_{p+1}^r(z) = z^{-(p+1)}A_{p+1}(z^{-1})$ and the reverse of (2.18) we obtain the following recursion

$$A_{p+1}^r(z) = z^{-1}A_p^r(z) - \gamma_{p+1}A_p(z). \quad (2.21)$$

In Appendix B the Levinson-Durbin recursion is treated in matrix form and also for a more general situation than in this section.

2.2 The Levinson and related algorithm.

Equation (2.20) and (2.21) may be combined into a 2×2 matrix equation

$$\begin{bmatrix} A_{p+1}(z) \\ A_{p+1}^r(z) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{p+1}z^{-1} \\ -\gamma_{p+1} & z^{-1} \end{bmatrix} \begin{bmatrix} A_p(z) \\ A_p^r(z) \end{bmatrix} \quad (2.22)$$

The recursion is initialized at $p = 0$ by setting

$$A_0(z) = A_0^r(z) = 1 \text{ and } E_0 = R(0) = \mathbb{E}[y_n^2], \quad (2.23)$$

assuming no prediction at all. The next algorithm realizes the *Levinson recursion*

step1 initialize at $p = 0$, using (2.23)

step2 at stage p , the filter $A_p(z)$ and the error E_p are available

step3 compute γ_{p+1} , using (2.11)

step4 determine $A_{p+1}(z)$, using (2.18), (2.19) or (2.22)

step5 update E_{p+1} , using (2.15)

step6 $p := p + 1$ and go to step2 until $p > M$

In step 3 and 4 p multiplications are needed. The number of multiplications for a M^{th} order predictor is in the order of M^2 .

In the Split-Levinson algorithm this number is halved. For notation purposes we will determine the p^{th} order predictor. The first line of (2.22) becomes now $A_p(z) = A_{p-1}(z) - \gamma_p z^{-1} A_{p-1}^r(z)$ and by setting $\gamma_p = -1$, we consider the polynomial $F_p(z)$ derived from the predictor polynomials

$$F_p(z) = \sum_{i=0}^p f_{p,i} z^{-i} = A_{p-1}(z) + z^{-1} A_{p-1}^r(z). \quad (2.24)$$

By construction, $F_p(z)$ is symmetric, that is $f_{p,0} = 1$, $f_{p,p} = -\gamma_p = 1$ and

$$f_{p,i} = f_{p,p-i} = a_{p-1,i} + a_{p-1,p-i} \text{ for } i = 1, 2, \dots, p-1. \quad (2.25)$$

Using (2.22), with p in stead of $p+1$, and the definition of $F_p(z)$ we find

$$\lambda_p F_p(z) = A_p(z) + A_p^r(z). \quad (2.26)$$

with

$$\lambda_p = 1 - \gamma_p. \quad (2.27)$$

Introducing the vector $\mathbf{f}_p = (f_{p,0}, f_{p,1}, \dots, f_{p,p})^T$ and using (B.1) through (B.7) we can write

$$\mathbf{R}_p \mathbf{f}_p = \frac{1}{\lambda_p} (\mathbf{e}_p + \mathbf{e}_p^r) = \frac{1}{\lambda_p} (\mathbf{E}_p, 0, \dots, 0, \mathbf{E}_p)^T, \quad (2.28)$$

and

$$\tau_p = \frac{\mathbf{E}_p}{\lambda_p} = \sum_{i=0}^p R(i) f_{p,i}. \quad (2.29)$$

Because of the symmetric nature of f_p , the quantity τ_p can be computed using only half of the terms in the above inner product :

if p is odd $\tau_p = \sum_{i=0}^{(p-1)/2} [R(i) + R(p-i)] f_{p,i}$,

if p is even $\tau_p = \sum_{i=0}^{p/2-1} [R(i) + R(p-i)] f_{p,i} + R(p/2) f_{p,p/2}$.

If we replace p by $p+1$ in (2.24) and eliminate $A_p(z)$ and $A_p^r(z)$ respectively with (2.26), we obtain

$$(1 - z^{-1}) A_p(z) = F_{p+1}(z) - \lambda_p z^{-1} F_p(z) \quad (2.30)$$

$$(1 - z^{-1}) A_p^r(z) = -F_{p+1}(z) + \lambda_p F_p(z). \quad (2.31)$$

Substituting (2.30) and (2.31) with the correct order into $A_p(z) = A_{p-1}(z) - \gamma_p z^{-1} A_{p-1}^r(z)$, we get the three-term recurrence relation

$$F_{p+1}(z) = (1 + z^{-1}) F_p(z) - \alpha_p z^{-1} F_{p-1}(z) \quad (2.32)$$

with

$$\alpha_p = \lambda_{p-1} (2 - \lambda_p) = \frac{\tau_p}{\tau_{p-1}}. \quad (2.33)$$

The last equation is found with (2.27),(2.15) and (2.29) in that order as follows

$$\alpha_p = \lambda_{p-1}(1 + \gamma_p) = \frac{\lambda_{p-1}}{\lambda_p}(1 - \gamma_p^2) = \frac{\lambda_{p-1}}{\lambda_p} \frac{E_p}{E_{p-1}} = \frac{\tau_p}{\tau_{p-1}}.$$

Appropriate initial conditions are given by

$$F_0(z) = 2, F_1(z) = 1 + z^{-1}, \tau_0 = R(0). \quad (2.34)$$

The *Split-Levinson algorithm* becomes as follow

step1 Initialize at $p = 0$ according to (2.34)

step2 at stage p , F_{p-1}, F_p and τ_{p-1} are available

step3 compute τ_p with (2.29), using half the terms

step4 compute α_p with (2.33)

step4a compute $\gamma_p = -1 + \frac{\alpha_p}{1 - \gamma_{p-1}}$

step5 compute F_{p+1} with (2.32), using half the number of coefficients

step6 $p := p + 1$ and go to step2 until $p > M$

step6a compute $E_M = \tau_M(1 - \gamma_M)$

The algorithm given above is specified for the output of E_M and the reflection coefficients $\gamma_1, \gamma_2, \dots, \gamma_M$, so the calculation of F_{M+1} is unnecessary. If the prediction coefficients are wanted in stead of the reflection coefficients we proceed as follow. From (2.30) with $z = 1$ and $p = M$ we can resolve

$$\lambda_M = \frac{F_{M+1}(1)}{F_M(1)} = \frac{\sum_{i=0}^{M+1} f_{M+1,i}}{\sum_{i=0}^M f_{M,i}}. \quad (2.35)$$

Also from (2.30) with $p = M$ we have

$$a_{M,i} = a_{M,i-1} + f_{M+1,i} - \lambda_M f_{M,i-1}, \quad i = 1, 2, \dots, M \quad (2.36)$$

with as initialization $a_{M,0} = 1$. The algorithm changes as follow:

Step4a is discarded, at step5 F_{M+1} is calculated and at step 6a the prediction coefficients are calculated from (2.36) with the use of (2.35).

2.3 Analysis and synthesis filters

The traditional way of implementing the analysis and synthesize filters of figure 1.2 is via transversal or tapped-delay-line filters with coefficients $\{a_{M0}, \dots, a_{MM}\}$. Because the coefficients can vary over a large range, the drawback of this implementation is that unstable filters can be obtained if coefficients are used with finite precision arithmetic or even with quantized values. Therefore the lattice structured filter, using the PARCOR's as coefficients, is introduced [11] [12].

From (2.2) we know that $e_p(n)$ is the prediction error of a p^{th} order predictor. It is the convolution of the filter's impulse response with the original data sequence y_n , or in the z -domain

$$E_p(z) = A_p(z)Y(z). \quad (2.37)$$

Now the backward prediction error is introduced in terms of the reverse of the prediction filter $A_p^*(z)$

$$R_p(z) = A_p^*(z)Y(z) = z^{-p} A_p(z^{-1})Y(z). \quad (2.38)$$

So the signal sequence $r_p(n)$ becomes

$$r_p(n) = \sum_{i=0}^p a_{p,i} y_{n-p+i} = y_{n-p} + a_{p,1} y_{n-p+1} + \dots + a_{p,p} y_n \quad (2.39)$$

and may be interpreted as the postdiction error in postdicting the value of y_{n-p} on basis of the p future samples $\{y_{n-p+1}, y_{n-p+2}, \dots, y_n\}$. It is easy to show that

$$\mathbb{E}[r_p(n)^2] = \mathbb{E}[e_p(n)^2], \quad (2.40)$$

thus the forward and the backward prediction error criteria are the same. Both methods give the same solution for the optimal filter coefficients. By multiplying both sides of (2.22) by $Y(z)$ we obtain

$$\begin{bmatrix} E_{p+1}(z) \\ R_{p+1}(z) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{p+1} z^{-1} \\ -\gamma_{p+1} & z^{-1} \end{bmatrix} \begin{bmatrix} E_p(z) \\ R_p(z) \end{bmatrix} \quad (2.41)$$

and in the time domain

$$\begin{aligned} e_{p+1}(n) &= e_p(n) - \gamma_{p+1} r_p(n-1) \\ r_{p+1}(n) &= r_p(n-1) - \gamma_{p+1} e_p(n) \end{aligned} \quad (2.42)$$

The initial conditions can be read from (2.23) and are

$$E_0(z) = R_0(z) = A_0(z)Y(z) = Y(z) \text{ and } e_0(n) = r_0(n) = y_n.$$

The whitening or analysis filter as a feedforward lattice filter is given in figure 2.1. From this filter a lattice predictor can be constructed as follows. From (2.41) we have

$$E_{p+1}(z) = E_p(z) - \gamma_{p+1} z^{-1} R_p(z) \quad (2.43)$$

If we iterate from $p = 0$ to $p = M - 1$, we have $E_1(z) = E_0(z) - \gamma_1 z^{-1} R_0(z) = Y(z) - \gamma_1 z^{-1} R_0(z)$, $E_2(z) = Y(z) - \{\gamma_1 z^{-1} R_0(z) + \gamma_2 z^{-1} R_1(z)\}$ until $E_M(z) = Y(z) - \sum_{p=1}^M \gamma_p z^{-1} R_{p-1}(z)$. If the last expression is compared with (1.6) we have $\hat{Y}(z) = \sum_{p=1}^M \gamma_p z^{-1} R_{p-1}(z)$ or after a transformation

$$\hat{y}(n) = \sum_{p=1}^M \gamma_p r_{p-1}(n-1). \quad (2.44)$$

The lattice predictor is also depicted in figure 2.1 For the construction of the synthesis filter we must realize that $e_n = e_M(n)$ is the input of the filter, while $y_n = e_0(n)$ corresponds with the output. So the signal $e_p(n)$ must be calculated from $e_{p+1}(n)$ and from (2.41) we get

$$\begin{aligned} e_p(n) &= e_{p+1}(n) + \gamma_{p+1} r_p(n-1) \\ r_{p+1}(n) &= r_p(n-1) - \gamma_{p+1} e_p(n) \end{aligned} \quad (2.45)$$

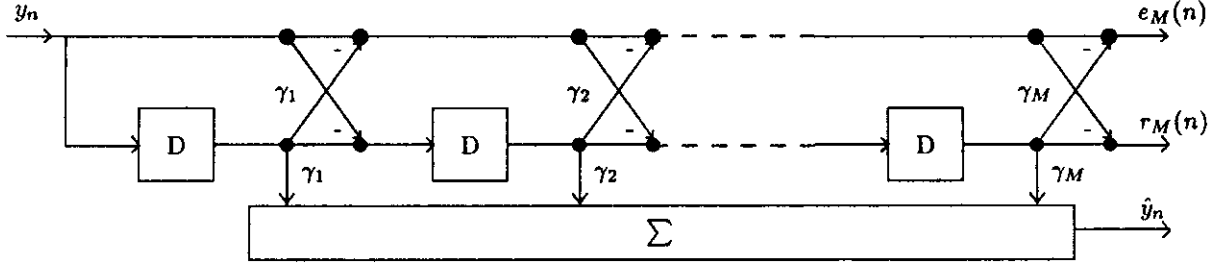


Figure 2.1: The feedforward lattice filter

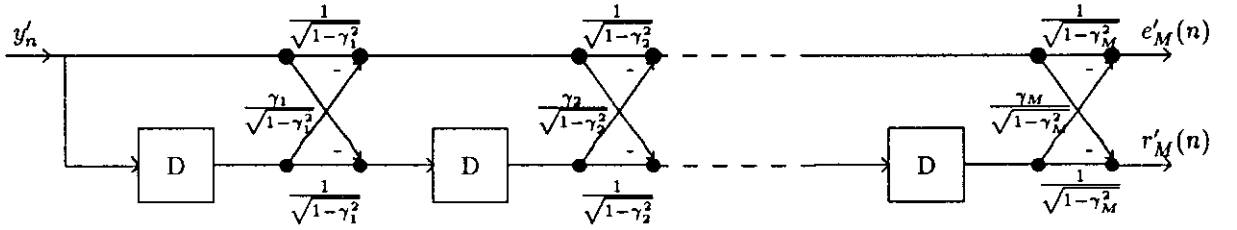


Figure 2.2: The normalized feedforward lattice filter

The synthesis or modeling filter as a feedback lattice filter is given in figure 2.3. In contrast with a transversal filter an increment of the order of the filter is just an addition of the appropriate number of sections. If the filter is implemented with distorted values of the PARCOR's, the filter stays minimal phase as long as these values satisfy $|\gamma_{p+1}| \leq 1$. By rearranging the terms of (2.45) we obtain the next expressions

$$\begin{aligned} e_p(n) &= e_{p+1}(n) + \gamma_{p+1}r_p(n-1) \\ r_{p+1}(n) &= -\gamma_{p+1}e_{p+1}(n) + (1 - \gamma_{p+1}^2)r_p(n-1), \end{aligned} \quad (2.46)$$

giving us the transmission-line filter of figure 2.4. This filter requires three multipliers per section, while the feedforward filter requires only two. By a suitable normalization of the formulas (2.46) we can obtain one of two goals, namely a filter with less multiplications or a filter with better finite precision arithmetic properties. Therefore the normalized forward and backward error signals $e'_p(n) = \frac{e_p(n)}{\delta_p}$ and $r'_p(n) = \frac{r_p(n)}{\delta_p}$ are introduced and (2.46) changes into

$$\begin{aligned} e'_p(n) &= \Delta_{p+1}e'_{p+1}(n) + \gamma_{p+1}r'_p(n-1) \\ r'_{p+1}(n) &= -\gamma_{p+1}e'_{p+1}(n) + (1 - \gamma_{p+1}^2)\Delta_{p+1}^{-1}r'_p(n-1), \end{aligned} \quad (2.47)$$

with $\Delta_{p+1} = \delta_{p+1}/\delta_p$. If we choose $\Delta_{p+1} = 1 + \gamma_{p+1}$ then (2.47) becomes

$$\begin{bmatrix} e'_p(n) \\ r'_{p+1}(n) \end{bmatrix} = \begin{bmatrix} 1 + \gamma_{p+1} & \gamma_{p+1} \\ -\gamma_{p+1} & 1 - \gamma_{p+1} \end{bmatrix} \begin{bmatrix} e'_{p+1}(n) \\ r'_p(n-1) \end{bmatrix} \quad (2.48)$$

This structure, given in the scattering matrix form, has four multipliers, but by combining terms this number reduces to one

$$\begin{aligned} e'_p(n) &= e'_{p+1}(n) + \gamma_{p+1}(e'_{p+1}(n) + r'_p(n-1)) \\ r'_{p+1}(n) &= r'_p(n-1) - \gamma_{p+1}(e'_{p+1}(n) + r'_p(n-1)). \end{aligned} \quad (2.49)$$

If we choose $\delta_p = E_p^{1/2}$, we normalize such that $e'_p(n)$ and $r'_p(n)$ have both unit energy and from (2.15) it follows that $\Delta_{p+1} = (1 - \gamma_{p+1}^2)^{1/2}$. So (2.47) becomes

$$\begin{bmatrix} e'_p(n) \\ r'_{p+1}(n) \end{bmatrix} = \mathbf{Q}(\gamma_{p+1}) \begin{bmatrix} e'_{p+1}(n) \\ r'_p(n-1) \end{bmatrix} \quad (2.50)$$

with

$$\mathbf{Q}(\gamma_{p+1}) = \begin{bmatrix} (1 - \gamma_{p+1}^2)^{1/2} & \gamma_{p+1} \\ -\gamma_{p+1} & (1 - \gamma_{p+1}^2)^{1/2} \end{bmatrix} \quad (2.51)$$

This normalized transmission-line lattice structure is given in figure 2.5. Notice that $\mathbf{Q}(\gamma_{p+1})$ is orthogonal i.e. $\mathbf{Q}(\gamma_{p+1})\mathbf{Q}^T(\gamma_{p+1}) = \mathbf{I}$. This garanties good numerical properties of the filter (no overflow oscillations and no limit cycles). Using the same normalization for the feedforward analysis filter, the structure of figure 2.2 is obtained. The scattering matrix for this situation is not orthogonal.

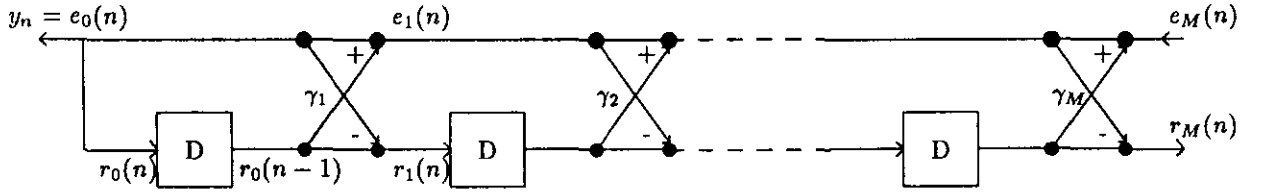


Figure 2.3: The feedforward lattice filter

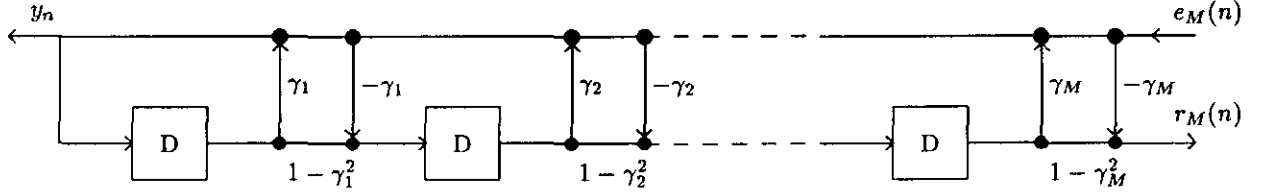


Figure 2.4: The transmission line filter

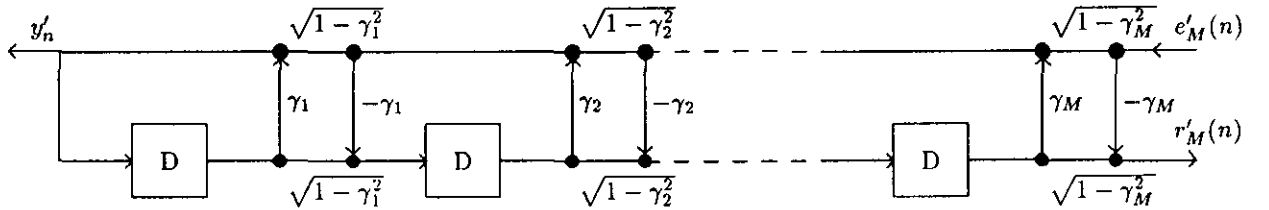


Figure 2.5: The normalized transmission line filter

Chapter 3

The Schur algorithms

The Schur algorithms are an efficient alternative to the Levinson algorithm and can be used to compute the set of reflection coefficients from the autocorrelation lags. The computation reduction is obtained only if parallel processing facilities are available.

In section 2.1 the forward gapped function $g_p(k)$ was introduced. Now we want to introduce the backward gapped function as

$$g_p^r(k) = \mathbb{E}[r_p(n)y_{n-k}].$$

By using (2.39) this becomes $g_p^r(k) = \sum_{i=0}^p a_{p,i} \mathbb{E}[y_{n-p+i}y_{n-k}] = \sum_{i=0}^p a_{p,i} R(-k + p - i)$. By a change of variables and by the use of the symmetry of the autocorrelation function, we obtain the next two gapped functions

$$g_p(k) = \sum_{i=0}^p a_{p,i} R(k - i) \quad g_p^r(k) = \sum_{i=0}^p a_{p,p-i} R(k - i). \quad (3.1)$$

The next three properties of these functions are important for our purpose

$$g_p^r(k) = g_p(p - k), \quad (3.2)$$

$$g_0(k) = g_0^r(k) = R(k), \quad (3.3)$$

$$g_p(0) = g_p^r(p) = E_p.$$

The properties can be found by the inspection of (3.1). From (2.8) we know that $g_p(k) = 0$ for $1 \leq k \leq p$, so with (3.2) we get $g_p^r(k) = 0$ for $0 \leq k \leq p - 1$. Because $a_{p,p-i}$ is the reverse of $a_{p,i}$, the z -transform of (3.1) is

$$G_p(z) = A_p(z)S_{yy}(z) \quad G_p^r(z) = A_p^r(z)S_{yy}(z),$$

which gives us also a relationship between $G_p^r(z)$ and $G_p(z)$

$$G_p^r(z) = z^{-p} A_p(z^{-1})S_{yy}(z^{-1}) = z^{-p} G_p(z^{-1}).$$

By the multiplication of (2.22) with $S_{yy}(z)$, we obtain the Schur recursion in the z -domain

$$\begin{bmatrix} G_{p+1}(z) \\ G_{p+1}^r(z) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{p+1}z^{-1} \\ -\gamma_{p+1} & z^{-1} \end{bmatrix} \begin{bmatrix} G_p(z) \\ G_p^r(z) \end{bmatrix} \quad (3.4)$$

and with the z -transform the Schur recursion for the two gapped functions

$$\begin{aligned} g_{p+1}(k) &= g_p(k) - \gamma_{p+1} g_p^r(k-1) \\ g_{p+1}^r(k) &= -\gamma_{p+1} g_p(k) + g_p^r(k-1) \end{aligned} \quad (3.5)$$

because $g_{p+1}(p+1) = 0$, γ_{p+1} is determined from (3.5) as

$$\gamma_{p+1} = \frac{g_p(p+1)}{g_p^r(p)}, \quad (3.6)$$

which is the same expression as (2.11). The next algorithm realizes the *Schur Recursion*

step1 initialize at $p = 0$, using (3.3) for $0 \leq k \leq M$

step2 at stage p , $g_p(k)$ and $g_p^r(k)$ are available for $p \leq k \leq M$

step3 compute γ_{p+1} , using (3.6)

step4 for $p+1 \leq k \leq M$ determine $g_{p+1}(k)$ and $g_{p+1}^r(k)$ with (3.5)

step5 $p := p+1$ and goto step2 until $p > M$

step6 make $E_M = g_M^r(M)$

During step3 the γ_{p+1} is determined as the ratio of two gapped functions, while at the Levinson algorithm p multiplications were needed for the same variable. At step4 $g_{p+1}(k)$ and $g_{p+1}^r(k)$ are calculated with each one multiplication and, because the necessary functions $g_p(k)$ and $g_p^r(k-1)$ are known, the maximal M multiplications can be done in parallel. So with M parallel processors the computational cost is of order M . In the Split-Schur algorithm the number of multiplication is reduced to one and a reduction of 50% is obtained.

As in section 2.2 we consider again a p^{th} order polynomial $G_p(z) = G_{p-1}(z) - \gamma_p z^{-1} G_{p-1}^r(z)$ and we also give γ_p the value -1. So

$$L_p(z) = G_{p-1}(z) + z^{-1} G_{p-1}^r(z)$$

can be compared with the function $F_p(z)$ from (2.24) and the result is

$$L_p(z) = F_p(z) S_{yy}(z). \quad (3.7)$$

The z -transform of this function becomes

$$l_p(k) = \sum_{i=0}^p f_{p,i} R(k-i). \quad (3.8)$$

The multiplication of (2.32) with $S_{yy}(z)$ and the use of (3.7) gives the three-term recurrence relation

$$\begin{aligned} L_{p+1}(z) &= (1 + z^{-1}) L_p(z) - \alpha_p z^{-1} L_{p-1}(z), \\ l_{p+1}(k) &= l_p(k) + l_p(k-1) - \alpha_p l_{p-1}(k-1). \end{aligned} \quad (3.9)$$

The variable α_p , given in (2.33), needs the variable τ_p , given in (2.29). But comparing (2.29) with (3.8), we see that $\tau_p = l_p(0) = l_p(p)$ and α_p becomes

$$\alpha_p = \frac{l_p(p)}{l_{p-1}(p-1)}. \quad (3.10)$$

The initialization conditions of (2.34) change into $L_0(z) = 2S_{yy}(z)$, $L_1(z) = (1 + z^{-1})S_{yy}(z)$ and $\tau_0 = R(0)$, or

$$\begin{aligned} l_0(k) &= 2R(k), \quad l_1(k) = R(k) + R(k-1), \quad \text{for } 1 \leq k \leq M \\ l_0(0) &= \tau_0 = R(0) \end{aligned} \quad (3.11)$$

The *Split-Schur algorithm* becomes as follow

step1 Initialize at $p = 0$ according to (3.11)

step2 at stage p , $l_{p-1}(k)$ for $p-1 \leq k \leq M$, $l_p(k)$ for $p \leq k \leq M$ and γ_{p-1} are available

step3 compute α_p with (3.10)

step4 compute $\gamma_p = -1 + \frac{\alpha_p}{1-\gamma_{p-1}}$

step5 determine $l_{p+1}(k)$ with (3.9) for $p+1 \leq k \leq M$

step6 $p := p + 1$ and goto step2 until $p > M$

step7 compute $E_M = l_M(M)(1 - \gamma_M)$

At step5 $l_{p+1}(k)$ can be calculated with one multiplication and with maximal M parallel processors, so a reduction of 50% is obtained compared with the Schur algorithm.

For both the Split-Levinson and the Split-Schur algorithm, γ_p was given a value equal to -1 in the functions $A_p(z) = A_{p-1}(z) - \gamma_p z^{-1} A_{p-1}^r(z)$ and $G_p(z) = G_{p-1}(z) - \gamma_p z^{-1} G_{p-1}^r(z)$ respectively. When γ_p is given the value 1, similar results can be obtained. This recursion with a fixed value for γ_p of $+1$ or -1 is also used for the determination of the line spectrum pairs (LSP). This will be the subject of the next section.

Chapter 4

The Line Spectrum Pairs (LSP)

The all-zero prediction filter or the corresponding all-pole synthesis filter can be described by the set of prediction coefficients $\{a_i\}$ or by the set of reflection coefficients $\{\gamma_i\}$. The Line spectrum Pairs (LSP) provide an alternative parameterization of the analysis and synthesis filter. In this chapter the LSP are defined, some properties are mentioned and three algorithms to determine the LSP are given.

As in section 2.2, we substitute in $A_{p+1}(z)$ a value of -1 for the variable γ_{p+1} to obtain the function $F_{p+1}(z)$, and also a value of 1 to obtain a function called $Q_{p+1}(z)$. So the next two functions appear

$$\begin{aligned} F_{p+1}(z) &= \sum_{i=0}^{p+1} f_{p+1,i} z^{-i} = A_p(z) + z^{-1} A_p^r(z) = A_p(z) + z^{-(p+1)} A_p(z^{-1}) \\ Q_{p+1}(z) &= \sum_{i=0}^{p+1} q_{p+1,i} z^{-i} = A_p(z) - z^{-1} A_p^r(z) = A_p(z) - z^{-(p+1)} A_p(z^{-1}). \end{aligned} \quad (4.1)$$

In the previous chapters p was a recursion parameter running from 1 to the order M of the filters. Here such a kind of recursion is impossible. So we take $F_{M+1}(z)$ and $Q_{M+1}(z)$ and these can be regarded as predictors of order $M+1$ obtained from $A_M(z)$ via for example the Levinson's algorithm by letting the $(M+1)^{th}$ reflection coefficient be -1 or $+1$, respectively. These polynomials have the following symmetry properties

$$\begin{aligned} f_{M+1,0} &= f_{M+1,M+1} = 1 \\ f_{M+1,i} &= f_{M+1,M+1-i} = a_{M,i} + a_{M,M+1-i} \\ q_{M+1,0} &= -q_{M+1,M+1} = 1 \\ q_{M+1,i} &= -q_{M+1,M+1-i} = a_{M,i} - a_{M,M+1-i} \\ q_{M+1,(M+1)/2} &= 0 \text{ for } M+1 \text{ is even.} \end{aligned} \quad (4.2)$$

The choice of $\gamma_{M+1} = \pm 1$ has as consequence that $E_{M+1} = (1 - \gamma_{M+1}^2) E_M = 0$ and if (B.1) and (B.2) are used it is easy to see that $R_{M+1} f_{M+1} = 0$ and $R_{M+1} q_{M+1} = 0$. Both f_{M+1} and q_{M+1} , vectors formed from the coefficients of the polynomials $F_{M+1}(z)$ and $Q_{M+1}(z)$, are eigenvectors of the matrix R_{M+1} with eigenvalues zero. Because only $R(0)$ to $R(M)$ are known, it remains to determine $R(M+1)$. Looking at (2.11) and (2.13) we see that

$$\gamma_{M+1} E_M = \epsilon_M = a_{M,0} R(M+1) + \dots + a_{M,M} R(1)$$

or

$$R(M+1) = \gamma_{M+1} E_M - \sum_{i=1}^M a_{M,i} R(M+1-i).$$

The LSP are determined by the roots or the zeros of the polynomials $F_{M+1}(z)$ and $Q_{M+1}(z)$. The LSP are also called Line Spectrum Frequencies (LSF) or Pisarenko frequencies [4] [5]. The zeros of $F_{M+1}(z)$ and $Q_{M+1}(z)$ are interlaced with each other and because of the symmetry, the roots are on the complex unit circle and appear as complex conjugate pairs, z_i and z_i^* [25] [28]. Therefore the roots can be combined as $e^{j\omega_i} + e^{-j\omega_i} = 2 \cos \omega_i$ and the ω_i 's are the LSP frequencies. If M is even, we can write

$$\begin{aligned} F_{M+1}(z) &= (1+z^{-1}) \prod_{i=1}^{M/2} (1 - z_i z^{-1})(1 - z_i^* z^{-1}) \\ &= (1+z^{-1}) \prod_{i=1}^{M'} (1 - 2 \cos \omega_i z^{-1} + z^{-2}), \end{aligned} \quad (4.3)$$

with $M' = M/2$. The polynomial $Q_{M+1}(z)$ becomes

$$Q_{M+1}(z) = (1-z^{-1}) \prod_{i=1}^{M'} (1 - 2 \cos \omega'_i z^{-1} + z^{-2}). \quad (4.4)$$

If we define $c_i = -2 \cos \omega_i$ and $c'_i = -2 \cos \omega'_i$, the analysis filter $A_M(z)$ can be recovered from the LSP as

$$\begin{aligned} A_M(z) &= \frac{1}{2} [F_{M+1}(z) + Q_{M+1}(z)] \\ &= \frac{1}{2} \left\{ \prod_{i=1}^{M'} (1 + c_i z^{-1} + z^{-2}) + \prod_{i=1}^{M'} (1 + c'_i z^{-1} + z^{-2}) \right\} \\ &\quad + z^{-1} \left\{ \prod_{i=1}^{M'} (1 + c_i z^{-1} + z^{-2}) - \prod_{i=1}^{M'} (1 + c'_i z^{-1} + z^{-2}) \right\}, \end{aligned}$$

which is depicted in Figure 4.1. For the synthesis filter $1/A_M(z)$ we use the approach of Figure 1.2. So the predictor $H(z) = 1 - A(z)$ must be determined in terms of c_i and c'_i . From (4.3) and from (4.4) we find

$$\begin{aligned} H(z) &= 1 - A_M(z) = \frac{1}{2} [1 - F_{M+1}(z) + 1 - Q_{M+1}(z)] \\ &= -\frac{1}{2} z^{-1} \left[\sum_{i=1}^{M'} (c_i + z^{-1}) \prod_{j=1}^{i-1} (1 + c_j z^{-1} + z^{-2}) + \prod_{i=1}^{M'} (1 + c_i z^{-1} + z^{-2}) \right. \\ &\quad \left. + \sum_{i=1}^{M'} (c'_i + z^{-1}) \prod_{j=1}^{i-1} (1 + c'_j z^{-1} + z^{-2}) - \prod_{i=1}^{M'} (1 + c'_i z^{-1} + z^{-2}) \right]. \end{aligned} \quad (4.5)$$

The predictor is shown in Figure 4.2.

To avoid the trigonometric storage or calculation of the coefficients c_i and c'_i , the Chebyshev polynomials are introduced [23]. These polynomials also offer a possibility to determine

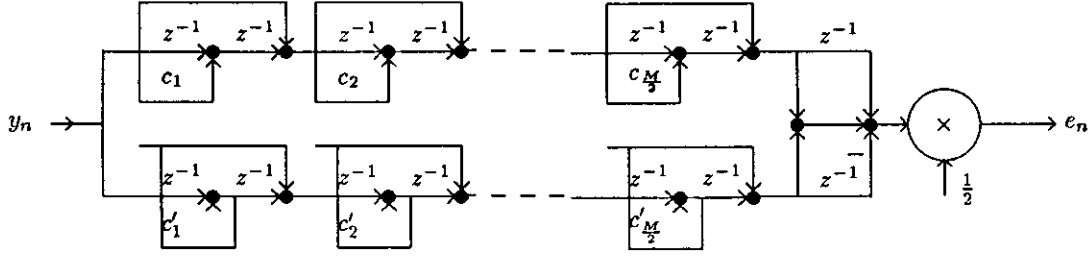


Figure 4.1: The LSP analysis lattice filter

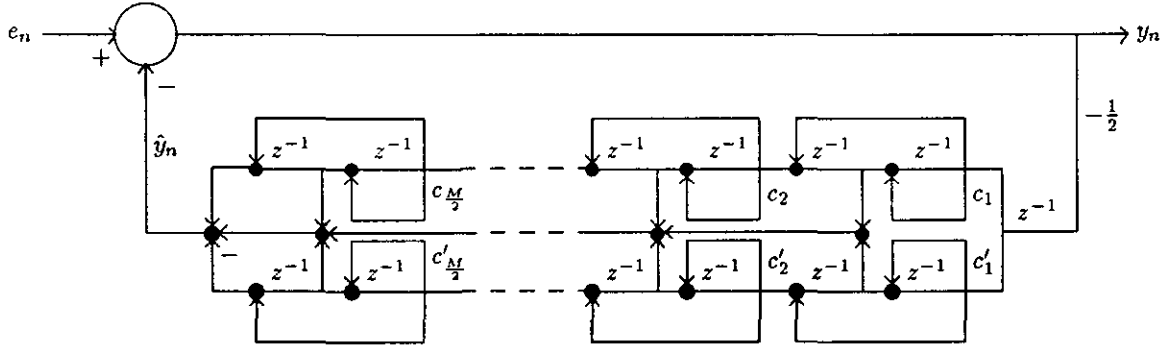


Figure 4.2: The LSP predictor

the LSF [10]. In stead of $F_{M+1}(z)$ from (4.1) and (4.3) and of $Q_{M+1}(z)$ from (4.1) and (4.4) two other polynomials are introduced.

$$\begin{aligned} F(z) &= \sum_{i=0}^M f_i z^{-i} = \frac{F_{M+1}(z)}{1+z^{-1}} \\ Q(z) &= \sum_{i=0}^M q_i z^{-i} = \frac{Q_{M+1}(z)}{1-z^{-1}} \end{aligned} \quad (4.6)$$

with

$$\begin{aligned} f_i &= \sum_{j=i+1}^{M+1} (-1)^{j-i-1} f_{M+1,j} \\ q_i &= \sum_{j=i+1}^{M+1} -q_{M+1,j} \end{aligned} \quad (4.7)$$

which have the symmetric property $f_i = f_{M-i}$ and $q_i = q_{M-i}$. A more efficient way to calculate the coefficients of $F(z)$ and $Q(z)$ is by using the following recursion. From $F_{M+1}(z) = (1+z^{-1})F(z)$ and from $Q_{M+1}(z) = (1-z^{-1})Q(z)$ we find $f_{M+1,i} = f_i + f_{i-1}$ and $q_{M+1,i} = q_i - q_{i-1}$, so the recursions become

$$f_0 = f_M = f_{M+1,0} = 1$$

$$\begin{aligned}
q_0 &= q_M = q_{M+1,0} = 1 \\
f_i &= f_{M-i} = f_{M+1,i} - f_{i-1}; \quad 1 \leq i \leq M' \\
q_i &= q_{M-i} = q_{M+1,i} + q_{i-1}; \quad 1 \leq i \leq M'.
\end{aligned} \tag{4.8}$$

Now we take only z values on the unit circle, $z = e^{j\omega}$, so the $F(z)$ from (4.6) becomes

$$F(e^{j\omega}) = e^{-j\omega M'} \sum_{i=0}^M f_i e^{j\omega(M'-i)} = e^{-j\omega M'} [2 \sum_{i=0}^{M'-1} f_i \cos \omega(M'-i) + f_{M'}]. \tag{4.9}$$

For the polynomial $Q(e^{j\omega})$ we have a same expression, but the f_i 's are replaced by the q_i 's. For M is *odd* equation (4.6) changes into

$$\begin{aligned}
F(z) &= \sum_{i=0}^M f_i z^{-i} = F_{M+1}(z) \\
Q(z) &= \sum_{i=0}^M q_i z^{-i} = \frac{Q_{M+1}(z)}{1 - z^{-2}},
\end{aligned} \tag{4.10}$$

and the recursion (4.8) changes into, with $M_1 = \frac{M+1}{2}$ and $M_2 = \frac{M-1}{2}$,

$$\begin{aligned}
f_0 &= f_{M+1} = f_{M+1,0} = 1 \\
q_0 &= q_{M-1} = q_{M+1,0} = 1 \\
q_1 &= q_{M-2} = q_{M+1,1} \\
f_i &= f_{M+1-i} = f_{M+1,i}; \quad 1 \leq i \leq M_1 \\
q_i &= q_{M-1-i} = q_{M+1,i} + q_{i-2}; \quad 2 \leq i \leq M_2.
\end{aligned} \tag{4.11}$$

The polynomials $F(e^{j\omega})$ and $Q(e^{j\omega})$ have now also the same expression as (4.9), but M' is replaced by M_1 and M_2 , respectively. If $x = \cos \omega$, then the k^{th} order Chebyshev polynomial $T_k(x)$ is defined as

$$T_k(x) = \cos k\omega = \cos k(\arccos x)$$

and from the trigonometric identity $\cos k\omega + \cos(k-2)\omega = 2 \cos \omega \cos(k-1)\omega$ the recursion

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \tag{4.12}$$

is obtained. The polynomials F and Q with the term $e^{-j\omega M'}$ removed, transform into

$$\begin{aligned}
F'(x) &= \sum_{k=0}^{M'} c_k T_k(x) \\
Q'(x) &= \sum_{k=0}^{M'} c'_k T_k(x)
\end{aligned} \tag{4.13}$$

with

$$\begin{aligned}
c_k &= 2f_{M'-k}; \quad 0 < k \leq M' \\
c_0 &= f_{M'} \\
c'_k &= 2q_{M'-k}; \quad 0 < k \leq M' \\
c'_0 &= q_{M'}
\end{aligned} \tag{4.14}$$

Applying the recursion of (4.12) for M' on $F'(x)$ we get

$$\begin{aligned} F'(x) &= c_0 + c_1 T_1(x) + \cdots + (c_{M'-2} - c_{M'}) T_{M'-2}(x) + (c_{M'-1} + 2x c_{M'}) T_{M'-1}(x) \\ &= c_0^1 + c_1^1 T_1(x) + \cdots + c_{M'-2}^1 T_{M'-2}(x) + c_{M'-1}^1 T_{M'-1}(x) \end{aligned}$$

with $c_j^1 = c_j$; $j = 0, \dots, M' - 3$, $c_{M'-2}^1 = c_{M'-2} - c_{M'}$, and $c_{M'-1}^1 = c_{M'-1} + 2x c_{M'}$. We now continue to apply the three-term recurrence formula to obtain

$$F'(x) = c_0^k + c_1^k T_1(x) + \cdots + c_{M'-k}^k T_{M'-k}(x) \quad (4.15)$$

with

$$\begin{aligned} c_j^k &= c_j^{k-1} = c_j; \quad j = 0, \dots, M' - (k + 2) \\ c_{M'-(k+1)}^k &= c_{M'-(k+1)}^{k-1} - c_{M'-(k-1)}^{k-1} \\ c_{M'-k}^k &= c_{M'-k}^{k-1} + 2x c_{M'-(k-1)}^{k-1} \end{aligned} \quad (4.16)$$

as long as $k \leq M' - 1$. If we take $b_k = c_{M'-k}^{M'-k}$ then we have from (4.16) for $k = M', M' - 1, \dots, 1$ the backward recurrence relationship

$$b_k = 2x b_{k+1} - b_{k+2} + c_k \quad (4.17)$$

with $b_{M'+1} = b_{M'+2} = 0$. For $k = M' - 1$ we have from (4.15)

$$\begin{aligned} F'(x) &= c_0^{M'-1} + c_1^{M'-1} T_1(x) \\ &= c_0^{M'-2} - c_2^{M'-2} + c_1^{M'-1} T_1(x) \\ &= c_0 - b_2 + b_1 x \\ &= \frac{c_0 + b_0 - b_2}{2} \end{aligned} \quad (4.18)$$

For the last equality in (4.18) we used the recursion from (4.17) for $k = 0$. So the values of $F'(x)$ and $Q'(x)$ for a certain value of x can be obtained easy by the backward recurrence (4.17) and by using (4.18). This calculation will be used later in this chapter to construct an efficient algorithm to find the roots of $F'(x)$ and $Q'(x)$.

In a coding system these M real valued roots are transmitted or recorded, so we need an efficient method to construct the synthesis filter from these x parameters. If the roots of $F'(x)$ are known and numbered $x_1, \dots, x_{M'}$ and if the K^{th} order polynomial in $T_k(x)$, $F'_K(x) = \sum_{k=0}^K c_k^K T_k(x)$, is constructed from K roots, then the $(K+1)^{th}$ polynomial $F'_{K+1}(x)$ is obtained by using root x_{K+1} as

$$\begin{aligned} F'_{K+1} &= 2(x - x_{K+1}) F'_K(x) \\ &= \sum_{k=-1}^{K+1} (c_{k-1}^K - 2x_{K+1} c_k^K + c_{k+1}^K) T_k(x) \end{aligned} \quad (4.19)$$

Here the recursion $2x T_k(x) = T_{k-1}(x) + T_{k+1}(x)$ is used again. So $F'(x)$ can be found from its roots with the recurrence relationship

$$c_k^{K+1} = c_{k-1}^K - 2x_{K+1} c_k^K + c_{k+1}^K \quad (4.20)$$

for $-1 \leq k \leq K+1$, and with $c_k^K = 0$ for $k < 0$ and for $k > K$, and $c_0^0 = 1$. While c_{-1}^{K+1} is the coefficient of $T_{-1}(x) = T_1(x)$, c_{-1}^{K+1} must be added to c_1^{K+1} to get the coefficient of $T_1(x)$. Once $F'(x)$ and $Q'(x)$ are reconstructed, it is easy to obtain $F_{M+1}(z)$, $Q_{M+1}(z)$ and the filter $A(z) = \frac{F_{M+1}(z) + Q_{M+1}(z)}{2}$.

Three methods for the LSP determination are described

1. A root finding algorithm for polynomials, such as the routine LAGUER or ZROOTS.
2. The roots of a polynomial are the eigenvalues of the companion matrix of the polynomial. The companion matrix of the polynomial $F_{M+1}(z) = \sum_{i=0}^{M+1} f_i z^i$ for example is

$$\begin{bmatrix} -f_M & -f_{M-1} & & -f_1 & -f_0 \\ 1 & 0 & & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & & 0 & 0 \\ 0 & 0 & & 1 & 0 \end{bmatrix}$$

Note the *positive* powers of z in the polynomial used in this example. If this companion matrix is called A , then x is an eigenvector of A and λ is an eigenvalue of A if $Ax = \lambda x$, or $[A - \lambda I]x = 0$. Because $\det[A - \lambda I] = 0$, it is easily checked that $\lambda^{M+1} + \sum_{i=0}^M f_i \lambda^i = 0$. Note that the first index of the coefficients of the polynomial is omitted and that $f_{M+1} = 1$. Because the matrix A is an Hessenberg matrix the eigenvalues can be determined by the routine HQR.

3. The roots of $F'(x)$ and $Q'(x)$ can be found by a linear search for a sufficient small interval in which the function value changes of sign. The search starts at $x = 1$ for a root of $F'(x)$ and goes backward with an interval of δ . If an interval is found with a sign change, the interval is successive bisectioned until the required precision of the root position is achieved. The midpoint of the interval is declared the first root of $F'(x)$ and is also used as the starting point for the search a root of $Q'(x)$. This procedure is repeated until M roots are determined. The value δ must be smaller than the minimum distance between two successive roots of $F'(x)$ or of $Q'(x)$. After N bisections of the interval of size δ the root precision becomes $\delta(\frac{1}{2})^{N+1}$, which value must be smaller than the minimum distance between a pair of roots, one of $F'(z)$ and one of $Q'(x)$. The *LSP algorithm* becomes now as follows

- step1** Determine $F(z)$ and $Q(z)$ from $F_{M+1}(z)$ and $Q_{M+1}(z)$ using (4.8) or (4.11).
- step2** Determine the coefficients $\{c_i\}$ of $F'(x)$ and the coefficients $\{c'_i\}$ of $Q'(x)$ with (4.14).
- step3** $x := 1$ and calculate the value of $F'(x)$ with (4.17) and (4.18).
- step4** $x := x - \delta$ and calculate $F'(x)$ if sign change then begin bisection N times for root x^f of $F'(x)$, $x := x^f$, calculate $Q'(x)$ and goto step5 end else goto step4.
- step5** $x := x - \delta$ and calculate $Q'(x)$ if sign change then begin bisection N times for root x^q of $Q'(x)$, $x := x^q$, calculate $F'(x)$ and goto step4 end else goto step5.
- step6** Repeat step4 and step5 until M roots are found.

The routines LAGUER, ZROOTS and HQR are from [21].

Example 1

This is a rather detailed example because we want to do a great deal of practising the calculations we have seen so far. Given are $R(0) = 8$, $R(1) = 4$ and $R(2) = -1$, so $M = 2$. We want to determine the roots of $F_3(z)$ and $Q_3(z)$.

Step 1 $\gamma_1 = R(1)/R(0) = 0.5$, $E_1 = (1 - \gamma_1^2)E_0 = (1 - \gamma_1^2)R(0) = 6$. $a_{1,0} = 1$, $a_{1,1} = -\gamma_1 = -0.5$.

Step 2 $\gamma_2 = (R(2) + a_{1,1}R(1))/E_1 = -0.5$, $E_2 = (1 - \gamma_2^2)E_1 = 4.5$. $a_{2,0} = 1$, $a_{2,1} = a_{1,1} - \gamma_2 a_{1,1} = -0.75$, $a_{2,2} = -\gamma_2 = 0.5$.

Step 3a Determination of $F_3(z)$ by assuming $\gamma_3 = -1$. $f_{3,0} = 1$, $f_{3,1} = a_{2,1} - \gamma_3 a_{2,2} = -0.25$, $f_{3,2} = a_{2,2} - \gamma_3 a_{2,1} = -0.25$, $f_{3,3} = -\gamma_3 = 1$. $R(3) = \gamma_3 E_2 - (a_{2,1}R(2) + a_{2,2}R(1)) = 4.5\gamma_3 - 2.75 = -7.25$.

The roots of $F_3(z)$ are found by $1 - 0.25z^{-1} - 0.25z^{-2} + z^{-3} = 0$ or $(1 + z^{-1})(1 - 1.25z^{-1} + z^{-2}) = 0$ and are $z_1 = -1$ and $z_{1,2} = (5 \pm j\sqrt{39})/8$. So $c_1 = -2 \cos \omega_1 = -2\frac{5}{8} = -\frac{5}{4}$.

Step 3b Determination of $Q_3(z)$ by assuming $\gamma_3 = 1$. $q_{3,0} = 1$, $q_{3,1} = a_{2,1} - \gamma_3 a_{2,2} = -1.25$, $q_{3,2} = a_{2,2} - \gamma_3 a_{2,1} = 1.25$, $q_{3,3} = -\gamma_3 = -1$. $R(3) = \gamma_3 E_2 - (a_{2,1}R(2) + a_{2,2}R(1)) = 4.5\gamma_3 - 2.75 = 1.75$. The roots of $Q_3(z)$ are found by $1 - 1.25z^{-1} + 1.25z^{-2} - z^{-3} = 0$ or $(1 - z^{-1})(1 - 0.25z^{-1} + z^{-2}) = 0$ and are $z_1 = 1$ and $z_{1,2} = (1 \pm j\sqrt{63})/8$. So $c'_1 = -2 \cos \omega'_1 = -2\frac{1}{8} = -\frac{1}{4}$.

Step 3c Determination of $A_M(z)$. $A_2(z) = \frac{1}{2}[F_3(z) + Q_3(z)] = \frac{1}{2}[(1 + z_{-1})(1 + c_1 z^{-1} + z^{-2})] = 1 - \frac{3}{4}z^{-1} + \frac{1}{2}z^{-2}$.

Step 4 By computer, the predictor (a) and the reflection coefficients (γ) can be checked with the routine Levinson, the γ -coefficients with the routines Split_Levinson, Schur or Split_Schur. These γ -coefficients can be transformed to a -coefficients with the routine StepUp. The routine Make.fq must be used to obtain the f and the q -coefficients, which can be fed into the routine Roots_zr, Roots_com or Roots_cheb to check the roots of the polynomials. A description of these routines, written in Pascal, can be found in appendix D.

Example 2

Here we take $M = 4$, and we assume that the analysis polynomial is $A(z) = 1 - 1.3z^{-1} + 0.7z^{-2} - 1.0z^{-3} + 0.8z^{-4}$. Then we obtain

$$\begin{aligned} F_{M+1}(z) &= 1 - 0.5z^{-1} - 0.3z^{-2} - 0.3z^{-3} - 0.5z^{-4} + z^{-5} \\ Q_{M+1}(z) &= 1 - 2.1z^{-1} + 1.7z^{-2} - 1.7z^{-3} + 2.1z^{-4} - z^{-5}. \end{aligned}$$

First we want to find the roots of $F'(z)$ and of $Q'(z)$. The $F(z)$ and $Q(z)$ polynomials of (4.6) are

$$\begin{aligned} F(z) &= 1 - 1.5z^{-1} + 1.2z^{-2} - 1.5z^{-3} + z^{-4} \\ Q(z) &= 1 - 1.1z^{-1} + 0.6z^{-2} - 1.1z^{-3} + z^{-4}. \end{aligned}$$

The values on the unit circle are

$$\begin{aligned} F(e^{j\omega}) &= e^{-2j\omega} [e^{2j\omega} - 1.5e^{j\omega} + 1.2 - 1.5e^{-j\omega} + e^{-2j\omega}] \\ Q(e^{j\omega}) &= e^{-2j\omega} [e^{2j\omega} - 1.1e^{j\omega} + 0.6 - 1.1e^{-j\omega} + e^{-2j\omega}] \end{aligned}$$

$$F'(x) = 2.0T_2(x) - 3.0T_1(x) + 1.2 = \sum_{k=0}^2 c_k T_k(x)$$

$$Q'(x) = 2.0T_2(x) - 2.2T_1(x) + 0.6 = \sum_{k=0}^2 c'_k T_k(x)$$

Both the polynomials (the solid curve is $F'(x)$) are shown in Figure 4.3. Because $b_4 = b_3 = 0$,

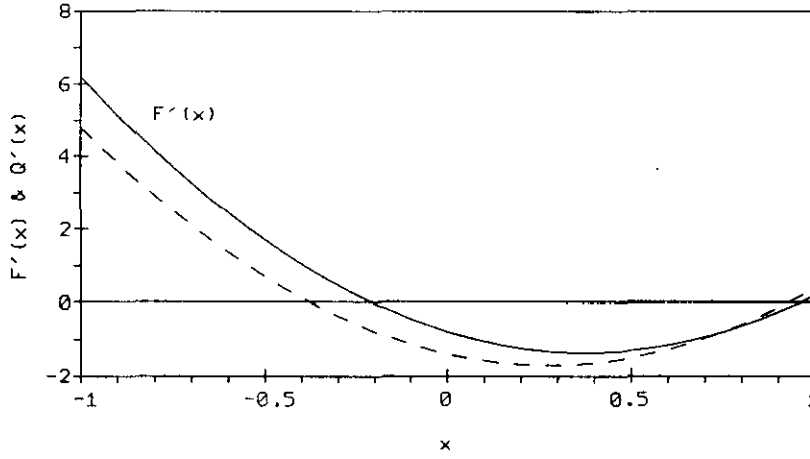


Figure 4.3: The polynomials $F'(x)$ and $Q'(x)$.

we have with (4.17) and (4.18)

$$\begin{aligned} b_2 &= 2xb_3 - b_4 + c_2 = 2 \\ b_1 &= 2xb_2 - b_3 + c_1 = 4x - 3 \\ b_0 &= 2x(4x - 3) - 2 + 1.2 \\ F'(x) &= \frac{b_0 - b_2 + c_0}{2} = 4x^2 - 3x - 0.8. \end{aligned}$$

In this case, the order M is small, the two roots of $F'(x)$ can be found analytical and are $x_1^f = 0.959$ and $x_2^f = -0.209$. In the same way the roots of $Q'(z)$ can be found as $x_1^q = 0.927$ and $x_2^q = -0.377$. The roots of $F_{M+1}(z)$ can be found as follows. For the root z_i we have the relation

$$z_i = e^{j\omega_i} = \cos \omega_i + j\sqrt{1 - \cos^2 \omega_i} = x_i^f + j\sqrt{1 - x_i^{f2}}$$

and we know that z_i^* is also a root. So we have the next five roots: -1 , $0.959 \pm j \times 0.283$ and $-0.209 \pm j \times 0.978$. Next we want to construct the polynomial $A(z)$ from the roots x_i^f and

x_i^q . The polynomial $F'(z)$ can be found as follows: for $K = 0$ we have from (4.20)

$$\begin{aligned} c_{-1}^1 &= c_{-2}^0 - 2x_1^f c_{-1}^0 + c_0^0 = c_0^0 = 1 \\ c_0^1 &= c_{-1}^0 - 2x_1^f c_0^0 + c_1^0 = -2x_1^f \\ c_1^1 &= c_0^0 - 2x_1^f c_1^0 + c_2^0 = c_0^0 = 1 \end{aligned}$$

and combining c_{-1}^1 and c_1^1 the result for $K = 0$ becomes

$$\begin{aligned} c_1^1 &= 2c_0^0 = 2 \\ c_0^1 &= -2x_1^f. \end{aligned}$$

For $K = 1$ we find with (4.20)

$$\begin{aligned} c_{-1}^2 &= c_{-2}^1 - 2x_2^f c_{-1}^1 + c_0^1 = c_0^1 \\ c_0^2 &= c_{-1}^1 - 2x_2^f c_0^1 + c_1^1 = -2x_2^f c_0^1 + c_1^1 \\ c_1^2 &= c_0^1 - 2x_2^f c_1^1 + c_2^1 = c_0^1 - 2x_2^f c_1^1 \\ c_2^2 &= c_1^1 - 2x_2^f c_2^1 + c_3^1 = c_1^1 \\ c_1^2 &= c_{-1}^2 + c_1^2 = 2c_0^1 - 2x_2^f c_1^1 \end{aligned}$$

and the substitution of the root values gives $c_0^2 = 1.2$, $c_1^2 = -3.0$ and $c_2^2 = 2.0$, the values of the coefficients of the $F'(x)$ polynomial. The coefficients of the polynomial $Q'(x)$ can be found in a similar way. From these coefficients the coefficients of $F_{M+1}(z)$ and $Q_{M+1}(z)$ can be found with (4.14) and (4.8) and then $A(z)$ with $A(z) = \frac{F_{M+1}(z) + Q_{M+1}(z)}{2}$.

Chapter 5

Other optimization criteria

In section 2.1 we found the optimal predictor using the minimum mean-square prediction error (MSE) as an optimization criterion. As a consequence, we got the set of linear equations (2.4) with the autocorrelation function defined as an ensemble average. If this autocorrelation function is not known a priori, the sample autocorrelation function can be used as an estimate. For a set of data $\{y_0, y_1, \dots, y_{N-1}\}$ this sample autocorrelation function is defined as

$$\hat{R}(p) = \frac{1}{N} \sum_{n=0}^{N-1-p} y_n y_{n+p}, \quad 0 \leq p \leq M. \quad (5.1)$$

Note that the normalization term $1/N$ drops out if (5.1) is substituted in (2.4). The autocorrelation matrix remains symmetric and Toeplitz.

An alternative approach is to replace the least mean square error by the least total square error (LSE) as the minimization criterion. So a time average is used instead of an ensemble average. For the same frame of data as mentioned above the total square error is defined as

$$E = \sum_{n \in N} e_M^2(n), \quad (5.2)$$

with N the range of n values taken into account. In principle the prediction error of a M^{th} order predictor can be determined for the values of n from 0 until $N - 1 + M$. But for the intervals $0 \leq n < M$ and $N - 1 < n \leq N - 1 + M$ a problem arises because not all the M data samples are available to make a good forward prediction. To include one or both intervals into N we need windows to make the sequence $\{y_{-M}, y_{-M+1}, \dots, y_{-1}\}$ or/and $\{y_N, y_{N+1}, \dots, y_{N-1+M}\}$ zero. So four distinct cases can be distinguished :

1. No windowing or the “covariance” method $N = \{M, M + 1, \dots, N - 1\}$.
2. Pre-windowing $N = \{0, 1, \dots, N - 1\}$.
3. Post-windowing $N = \{M, M + 1, \dots, N - 1 + M\}$.
4. Full-windowing or the “autocorrelation” method $N = \{0, 1, \dots, N - 1 + M\}$.

The unwindowed case and the windowed case has been mentioned covariance and autocorrelation methods respectively. This terminology is based on the historical usage in speech processing. It should be emphasized that the terms covariance method and autocorrelation

method are *not* based on the standard statistical definitions: The covariance function is the correlation function with the means removed. If the next two vectors

$$\begin{aligned} \mathbf{a}^T &= (1, a_{M,1}, \dots, a_{M,M}) \\ \mathbf{e}^T &= (e_M(0), e_M(1), \dots, e_M(N-1+M)), \end{aligned} \quad (5.3)$$

and a $(N+M) \times (M+1)$ matrix of data samples

$$\mathbf{Y} = \begin{bmatrix} y_0 & & & & \\ y_1 & y_0 & & & \\ \vdots & \vdots & & & \\ y_M & y_{M-1} & \cdots & & y_0 \\ \vdots & \vdots & & & \vdots \\ y_n & y_{n-1} & \cdots & & y_{n-M} \\ \vdots & \vdots & & & \vdots \\ y_{N-1} & y_{N-2} & \cdots & & y_{N-1-M} \\ & y_{N-1} & & & y_{N-2-M} \\ & & & & \vdots \\ & & & & y_{N-1} \end{bmatrix}$$

are introduced, the set of residuals can be written in a matrix form

$$\mathbf{e} = \mathbf{Y}\mathbf{a}.$$

The vector \mathbf{a} that gives the least total square error can be found from the equation

$$[\mathbf{Y}^T \mathbf{Y}] \mathbf{a} = (\mathbf{E}_m, 0, \dots, 0)^T,$$

where \mathbf{E}_m is the minimum value of (5.2) or the minimum of $\|\mathbf{e}\|^2$. The $(M+1) \times (M+1)$ matrix $\mathbf{R} = \mathbf{Y}^T \mathbf{Y}$ is Toeplitz *only* for the full window situation. The entries of the matrix are in this case the same as these from (5.1) without the normalization factor. If less windowing is applied, greater parts of \mathbf{e} and \mathbf{Y} are omitted and the matrix \mathbf{R} becomes less Toeplitz. In all the four cases, from full untill no windowing, the matrix \mathbf{R} is symmetric.

A third approach is the method where the sum of the forward and backward squared prediction error is minimized without windowing. Burg minimized these sum subject to the constraint that the prediction coefficients satisfy the Levinson-Durbin recursion. In the Marple least square algorithm the Levinson-Durbin recursion constraint is removed [15] [16].

5.1 The autocorrelation method

In this case the least total energy of (5.2) becomes, for a p^{th} order model,

$$\begin{aligned} E &= \sum_{n=0}^{N-1-p} e_p^2(n) \\ &= \sum_{n=0}^{N-1-p} (y_n + \sum_{i=1}^p a_{p,i} y_{n-i})^2 \end{aligned}$$

$$\begin{aligned}
&= \sum_{n=0}^{N-1} y_n^2 + 2 \sum_{i=1}^p a_{p,i} \sum_{n=0}^{N-1+i} y_n y_{n-i} + \sum_{i=1}^p \sum_{j=1}^p a_{p,i} a_{p,j} \sum_{n=i}^{N-1+j} y_{n-i} y_{n-j} \\
&= \hat{R}(0) + 2 \sum_{i=1}^p a_{p,i} \hat{R}(-i) + \sum_{i=1}^p \sum_{j=1}^p a_{p,i} a_{p,j} \hat{R}(i-j).
\end{aligned} \tag{5.4}$$

Here the definition of (5.1) is used and the relation

$$\hat{R}(-i) = \sum_{n=0}^{N-1+i} y_n y_{n-i} = \sum_{n=i}^{N-1} y_n y_{n-i} = \hat{R}(i).$$

To obtain the minimum energy the prediction coefficients are choosen according to

$$\sum_{j=0}^p a_{p,j} \hat{R}(i-j) = -\hat{R}(-i), \quad i = 1, 2, \dots, p \tag{5.5}$$

or

$$\begin{aligned}
\sum_{j=0}^p a_{p,j} \hat{R}(i-j) &= 0, \quad i = 1, 2, \dots, p \\
E_p &= \sum_{i=0}^p a_{p,i} \hat{R}(i),
\end{aligned} \tag{5.6}$$

where the last expression is for the minimal LSE. The equations of (5.6) can be compared with (2.4) and (2.5) and also combined into the Yule-Walker matrix equation of (2.6), but using the sample autocorrelation functions in stead of the ensemble averages. Because this autocorrelation matix is symmetric Toeplitz, the Levinson-Durbin recursion holds and all the algorithms found untill here are valid for the autocorrelation method.

Although the Cholesky decomposition (see Appendix C) is a computational inefficient method to find the prediction coefficients in the full-windowed situation, the usage of it gives us a further insight into the reflection coefficients. If the equations (5.5) are written as the matrix equation

$$\mathbf{R}\mathbf{a} = \mathbf{r}$$

and we use the Cholesky decomposition with the upper triangular matrix \mathbf{U} with one's on the diagonal, the second back substitution

$$\mathbf{U}\mathbf{a} = \mathbf{g}' \tag{5.7}$$

gives us a vector \mathbf{g}' containing the negative values of the reflection coefficients. If γ'_i is defined as $\gamma'_i = -\gamma_i$; $1 \leq i \leq p$, the Levinson-Durbin recursion (2.19) becomes

$$\begin{aligned}
a_{p,i} &= a_{p-1,i} + \gamma'_p a_{p-1,p-i}; \quad 1 \leq i < p \\
a_{p,p} &= \gamma'_p.
\end{aligned} \tag{5.8}$$

If we want to express the γ' 's into the prediction coefficients of only the p^{th} order predictor, $a_{p,i}$, we find

$$\gamma'_p = a_{p,p}$$

$$\begin{aligned}
\gamma'_{p-1} &= a_{p-1,p-1} = a_{p,p-1} - \gamma'_p a_{p-1,1} = a_{p,p-1} - a_{p-1,1} a_{p,p} \\
\gamma'_{p-2} &= a_{p,p-2} - a_{p-2,1} a_{p,p-1} + a_{p-1,2} (a_{p-1,1} - 1) a_{p,p} \\
\gamma'_{p-i} &= a_{p,p-i} + \sum_{j=i+1}^0 c_{p-i,j} a_{p,p-j} \\
\gamma'_i &= a_{p,i} + \sum_{j=i+1}^p c_{i,j} a_{p,j}.
\end{aligned} \tag{5.9}$$

The last line of (5.9) is equal to (5.7), so the second back substitution gives us the reflection coefficients with a minus sign.

5.2 The covariance method

If no windowing is applied, the vector of (5.3) reduces to

$$\mathbf{e}^T = (e_M(M), e_M(M+1), \dots, e_M(N-1)),$$

the $(N-M) \times (M+1)$ data matrix becomes

$$\mathbf{Y} = \begin{bmatrix} y_M & y_{M-1} & \cdots & y_0 \\ \vdots & \vdots & & \vdots \\ y_n & y_{n-1} & \cdots & y_{n-M} \\ \vdots & \vdots & & \vdots \\ y_{N-1} & y_{N-2} & \cdots & y_{N-1-M} \end{bmatrix}$$

and the entries of the covariance matrix $\mathbf{R} = \mathbf{Y}^T \mathbf{Y}$ are given by

$$R_{i,j} = \sum_{n=M}^{N-1} y_{n-i} y_{n-j}, \quad 0 \leq i, j \leq M. \tag{5.10}$$

It is easy to see that the matrix is symmetric, $R_{j,i} = R_{i,j}$, and that, for $1 \leq i, j \leq M$, the next recursive relation holds

$$R_{i,j} = R_{i-1,j-1} + y_{M-i} y_{M-j} - y_{N-i} y_{N-j}. \tag{5.11}$$

In the same manner as in the previous section the LSE can be written as

$$\begin{aligned}
E &= \sum_{n=M}^{N-1} e_M^2(n) \\
&= \sum_{n=M}^{N-1} (y_n + \sum_{i=1}^M a_{M,i} y_{n-i})^2 \\
&= \sum_{n=M}^{N-1} y_n^2 + 2 \sum_{i=1}^M a_{M,i} \sum_{n=M}^{N-1} y_n y_{n-i} + \sum_{i=1}^M \sum_{j=1}^M a_{M,i} a_{M,j} \sum_{n=M}^{N-1} y_{n-i} y_{n-j} \\
&= R_{0,0} + 2 \sum_{i=1}^M a_{M,i} R_{0,i} + \sum_{i=1}^M \sum_{j=1}^M a_{M,i} a_{M,j} R_{i,j},
\end{aligned} \tag{5.12}$$

where the definition of (5.10) for $R_{i,j}$ is used, and the optimum prediction coefficients are obtained by solving the equations

$$\sum_{j=1}^M a_{M,j} R_{i,j} = -R_{i,0}; \quad i = 1, 2, \dots, M. \quad (5.13)$$

The Yule-Walker equation contains now the next two expressions

$$\sum_{j=0}^M a_{M,j} R_{i,j} = 0, \quad i = 1, 2, \dots, M.$$

$$\mathbf{E}_M = \sum_{i=0}^M a_{M,i} R_{0,i}. \quad (5.14)$$

If we write the equations (5.13) in matrix form and because the matrix \mathbf{R} , containing the covariance functions, is symmetric we use the first Cholesky method to find the prediction coefficients (see appendix C)

$$\mathbf{R}\mathbf{a} = \mathbf{L}\mathbf{L}^T \mathbf{a} = \mathbf{r} = (-R_{0,1}, -R_{0,2}, \dots, -R_{0,M})^T.$$

Encouraged by the relation between the negative values of the reflection coefficients , \mathbf{g}' , and the prediction coefficients , \mathbf{a} , given by (5.7), we define here the (generalized) reflection coefficients as $\mathbf{U}\mathbf{a} = \mathbf{g}'$ or

$$\mathbf{U}(-\mathbf{a}) = -\mathbf{g}' = \mathbf{g} = (\gamma_1, \gamma_2, \dots, \gamma_M)^T. \quad (5.15)$$

The upper triangular matrix \mathbf{U} has one's on the diagonal, so we use the second Cholesky method to solve the matrix equation

$$\mathbf{R}\mathbf{a} = \mathbf{L}\mathbf{U}\mathbf{a} = -\mathbf{r} = (R_{0,1}, R_{0,2}, \dots, R_{0,M})^T. \quad (5.16)$$

The first back substitution gives the values of the (generalized) reflection coefficients

$$\mathbf{L}\mathbf{g} = \mathbf{r},$$

while the prediction coefficients can be obtained by the second back substitution (5.15). If the (generalized) reflection coefficients are used as the transmitted parameters of some speech coding system, they are not sufficient to determine the filter $A(z)$. Because the Levinson-Durbin relation is not valid, the second back substitution must be used. This implies that the upper triangular matrix and thus the covariance coefficients must be known at the synthesizer. But sometimes [27] the Levinson-Durbin recursion is used to obtain the \mathbf{a} -parameters and so $A(z)$ polynomial. This method is not a theoretical correct one, but gives insignificant differences.

5.3 The Burg algorithm

From (2.2) and from (2.39) the errors of a p^{th} order forward and backward prediction, $e_p(n)$ and $r_p(n)$, are

$$e_p(n) = \sum_{i=0}^p a_{p,i} y_{n-i}$$

$$r_p(n) = \sum_{i=0}^p a_{p,i} y_{n-p+i}. \quad (5.17)$$

By using the Levinson-Durbin recursion from (2.19)

$$\begin{aligned} a_{p,i} &= a_{p-1,i} - \gamma_p a_{p-1,p-i}, 1 \leq i \leq p-1 \\ a_{p,p} &= -\gamma_p \end{aligned} \quad (5.18)$$

we find

$$\begin{aligned} e_p(n) &= e_{p-1}(n) - \gamma_p r_{p-1}(n-1) \\ r_p(n) &= r_{p-1}(n-1) - \gamma_p e_{p-1}(n). \end{aligned} \quad (5.19)$$

Now the sum of the forward and backward squared prediction errors without windowing,

$$E_p = \sum_{n=p}^{N-1} \{e_p^2(n) + r_p^2(n)\}, \quad (5.20)$$

is minimized such that the Levinson-Dubin recursion (5.19) holds. The substitution of (5.19) into (5.20) gives

$$E_p = D_p \gamma_p^2 - 2N_p \gamma_p + D_p, \quad (5.21)$$

with D_p and N_p defined as

$$\begin{aligned} N_p &= 2 \sum_{n=p}^{N-1} e_{p-1}(n) r_{p-1}(n-1) \\ D_p &= \sum_{n=p}^{N-1} \{e_{p-1}^2(n) + r_{p-1}^2(n-1)\}. \end{aligned} \quad (5.22)$$

The optimal value of γ_p is obtained if the derivative of (5.21) with respect to γ_p , $\frac{\partial E_p}{\partial \gamma_p}$, is equal to zero. This gives

$$\gamma_p = \frac{N_p}{D_p} = \frac{2 \sum_{n=p}^{N-1} e_{p-1}(n) r_{p-1}(n-1)}{\sum_{n=p}^{N-1} \{e_{p-1}^2(n) + r_{p-1}^2(n-1)\}}. \quad (5.23)$$

For the denominator D_p of (5.23) the recursive relation

$$D_p = (1 - \gamma_{p-1}^2) D_{p-1} - e_{p-1}^2(p-1) - r_{p-1}^2(N-1) \quad (5.24)$$

can be found by inserting the equations (5.19), for order $p-1$, into the expression for D_p of (5.22) and by realizing that

$$\begin{aligned} \gamma_{p-1} &= \frac{2 \sum_{n=p-1}^{N-1} e_{p-2}(n) r_{p-2}(n-1)}{\sum_{n=p-1}^{N-1} \{e_{p-2}^2(n) + r_{p-2}^2(n-1)\}} \\ &= \frac{2 \sum_{n=p-1}^{N-1} e_{p-2}(n) r_{p-2}(n-1)}{D_{p-1}}. \end{aligned}$$

One has to observe that by using (5.18) the Levinson-Durbin recursion is used to obtain stable prediction filters with poles within the unit circle. This is also the reason that the relation $E_p = (1 - \gamma_p^2) E_{p-1}$ holds for all orders from 1 to M .

The *Burg algorithm* becomes now as follows

step1 initialize at $p = 0$, $e_0(n) = r_0(n) = y_n$ for $0 \leq n \leq N - 1$
step2 at stage p , $e_{p-1}(n)$ and $r_{p-1}(n)$ are available for $p - 1 \leq n \leq N - 1$
step3 compute γ_p using (5.23) and (5.24)
step4 compute $e_p(n)$ and $r_p(n)$ using (5.19) for $p \leq n \leq N - 1$
step5 $p := p + 1$ and goto step2 until $p > M$

The a-parameters, the prediction filter coefficients, that can be obtained from the γ -parameters of (5.23) are also used in a power spectrum estimation method called the Maximum Entropy Method (MEM) [3]. So the algorithm Burg, given in appendix D, is nearly equal to the routine MEMCOF of [21]. The routine EVLMEM of [21] realizes the conversion of the a-parameters into a power spectrum estimation.

In the previous method the minimum of E_p was found for γ_p with fixed values of $\gamma_{p-1}, \dots, \gamma_1$. Now we shall minimize E_p for γ_p and γ_{p-1} simultaneously [1]. The minimal energy for the optimal value of γ_p is obtained from (5.21) and (5.23) and becomes

$$E_p = D_p - \frac{N_p^2}{D_p}. \quad (5.25)$$

But D_p and N_p are both functions of γ_{p-1} , so the nominator of the derivative of (5.25) with respect to γ_{p-1} becomes

$$\begin{aligned} S(\gamma_{p-1}) &= D_p^2 D'_p - 2N_p D_p N'_p + N_p^2 D'_p \\ &= \sum_{i=0}^d s_i \gamma_{p-1}^i, \end{aligned} \quad (5.26)$$

where the function S is also written as a polynomial. So the problem of finding those values of γ_{p-1} for which the function S is zero becomes a root finding problem for polynomials. If N_p and D_p are written as polynomials

$$\begin{aligned} N_p &= n_0 + n_1 \gamma_{p-1} + n_2 \gamma_{p-1}^2 \\ D_p &= d_0 + d_1 \gamma_{p-1} + d_2 \gamma_{p-1}^2, \end{aligned} \quad (5.27)$$

the constants n_0, \dots, n_2 and d_0, \dots, d_2 can be found by plugging the expressions for $e_{p-1}(n)$ and $r_{p-1}(n-1)$, obtained from (5.19), into (5.22). The result of this operation for $p \geq 2$ is

$$\begin{aligned} d_0 &= \sum_{n=p}^{N-1} \{e_{p-2}^2(n) + r_{p-2}^2(n-2)\} \\ d_1 &= -2 \sum_{n=p}^{N-1} \{e_{p-2}(n)r_{p-2}(n-1) + e_{p-2}(n-1)r_{p-2}(n-2)\} \\ d_2 &= \sum_{n=p}^{N-1} \{e_{p-2}^2(n-1) + r_{p-2}^2(n-1)\} \\ n_0 &= 2 \sum_{n=p}^{N-1} e_{p-2}(n)r_{p-2}(n-2) \end{aligned}$$

$$\begin{aligned}
n_1 &= -2 \sum_{n=p}^{N-1} \{e_{p-2}(n)e_{p-2}(n-1) + r_{p-2}(n-1)r_{p-2}(n-2)\} \\
n_2 &= 2 \sum_{n=p}^{N-1} e_{p-2}(n-1)r_{p-2}(n-1).
\end{aligned} \tag{5.28}$$

If these polynomials (5.27) are used to obtain the polynomial of (5.26) the constants s_0, \dots, s_5 are

$$\begin{aligned}
s_0 &= d_1(d_0^2 + n_0^2) - 2d_0n_0n_1 \\
s_1 &= 2d_0(d_1^2 - n_1^2) + 2d_2(d_0^2 + n_0^2) - 4d_0n_0n_2 \\
s_2 &= d_1(d_1^2 - n_1^2) + 6d_0(d_1d_2 - n_1n_2) + 2n_0(d_2n_1 - d_1n_2) \\
s_3 &= 4d_1(d_1d_2 - n_1n_2) + 4d_0(d_2^2 - n_2^2) \\
s_4 &= d_1(5d_2^2 - 3n_2^2) - 2d_2n_1n_2 \\
s_5 &= 2d_2^2(d_2^2 - n_2^2).
\end{aligned} \tag{5.29}$$

The *Burg2 algorithm* becomes now as follows

step1 initialize at $p = 0$, $e_0(n) = r_0(n) = y_n$ for $0 \leq n \leq N - 1$

step2 at stage p , $e_{p-2}(n)$ and $r_{p-2}(n)$ are available for $p \leq n \leq N - 1$

step3 compute $n_0, \dots, n_2, d_0, \dots, d_2$ with (5.28)

step4 compute the polynomial (5.26) with (5.29) and find the roots with for example Roots.com. The real root is γ_{p-1} .

step5 compute γ_p using (5.27) and (5.23)

step6 compute $e_{p-1}(n)$ and $r_{p-1}(n)$ using (5.19) for $p - 1 \leq n \leq N - 1$ and $e_p(n)$ and $r_p(n)$ for $p \leq n \leq N - 1$

step7 $p := p + 2$ and goto step2 until $p > M$

5.4 The Marple algorithm

For the Marple algorithm we obtain the optimal values of the prediction coefficients by making $\frac{\partial E_p}{\partial a_{p,i}}$ equal to zero, for $1 \leq i \leq p$, and with E_p given in (5.20). If we substitute (5.17) into (5.20), we obtain the next relations

$$\begin{aligned}
E_p &= \sum_{n=p}^{N-1} \left[\sum_{i=0}^p \sum_{j=0}^p a_{p,i} y_{n-i} y_{n-j} a_{p,j} + \sum_{i=0}^p \sum_{j=0}^p a_{p,i} y_{n-p+i} y_{n-p+j} a_{p,j} \right] \\
&= \sum_{i=0}^p \sum_{j=0}^p a_{p,i} a_{p,j} \sum_{n=p}^{N-1} [y_{n-i} y_{n-j} + y_{n-p+i} y_{n-p+j}] \\
&= \sum_{i=0}^p \sum_{j=0}^p a_{p,i} a_{p,j} R_{i,j},
\end{aligned} \tag{5.30}$$

where in this section $R_{i,j}$ is defined as

$$R_{i,j} = \sum_{n=p}^{N-1} [y_{n-i}y_{n-j} + y_{n-p+i}y_{n-p+j}] \quad (5.31)$$

for $0 \leq i, j \leq p$. Differentiating (5.30) with respect to each coefficient $a_{p,i}$ gives

$$2 \sum_{j=0}^p a_{p,j} R_{i,j} = 0, \quad 1 \leq i \leq p \quad (5.32)$$

and the minimal value of (5.30) becomes

$$E_p = \sum_{j=0}^p a_{p,j} R_{0,j}. \quad (5.33)$$

The expressions (5.32) and (5.33) can be combined into a $(p+1) \times (p+1)$ matrix equation

$$\mathbf{R}_p \mathbf{a}_p = \mathbf{e}_p. \quad (5.34)$$

Because the elements $R_{i,j}$ from (5.31) are the entries of the matrix \mathbf{R}_p , these matrix is the sum of two matrices. The first one can be obtained, as in section 3.1 but now for a prediction order p in stead of M , from the $(N-p) \times (p+1)$ data matrix \mathbf{Y} as $\mathbf{Y}^T \mathbf{Y}$. The second one can be obtained in the same way from the reversed data matrix

$$\mathbf{Y}_r = \begin{bmatrix} y_0 & \cdots & y_{p-1} & y_p \\ \vdots & & \vdots & \vdots \\ y_{n-p} & \cdots & y_{n-1} & y_n \\ \vdots & & \vdots & \vdots \\ y_{N-1-p} & \cdots & y_{N-2} & y_{N-1} \end{bmatrix}$$

as $\mathbf{Y}_r^T \mathbf{Y}_r$. So \mathbf{R}_p is the sum of two Toeplitz data matrix products. It is easy to see that \mathbf{R}_p is symmetric, $R_{i,j} = R_{j,i}$ or $\mathbf{R}_p^T = \mathbf{R}_p$, and persymmetric, $R_{i,j} = R_{p-i,p-j}$. If two additional prediction error energy terms

$$\begin{aligned} E'_p &= \sum_{n=p}^{N-2} \{e_p^2(n+1) + r_p^2(n)\} \\ E''_p &= \sum_{n=p}^{N-2} \{e_p^2(n) + r_p^2(n+1)\} \end{aligned} \quad (5.35)$$

are minimized in a manner similar to that used for E_p , the next expressions, comparable to (5.34) can be obtained

$$\mathbf{R}'_p \mathbf{a}'_p = \mathbf{e}'_p \quad (5.36)$$

$$\mathbf{R}''_p \mathbf{a}''_p = \mathbf{e}''_p. \quad (5.37)$$

The entries of the matrices \mathbf{R}'_p and \mathbf{R}''_p are

$$\begin{aligned} R'_{i,j} &= \sum_{n=p}^{N-2} y_{n+1-i} y_{n+1-j} + y_{n-p+i} y_{n-p+j} \\ R''_{i,j} &= \sum_{n=p}^{N-2} y_{n-i} y_{n-j} + y_{n+1-p+i} y_{n+1-p+j} \end{aligned} \quad (5.38)$$

for $0 \leq i, j \leq p$ and are related with (5.31) as follows

$$\begin{aligned} R'_{i,j} &= R_{i,j} - y_{p-i} y_{p-j} - y_{N-1-p+i} y_{N-1-p+j} \\ R''_{i,j} &= R_{i,j} - y_i y_j - y_{N-1-i} y_{N-1-j}. \end{aligned} \quad (5.39)$$

Because of the persymmetric relation, $R_{i,j} = R_{p-i,p-j}$, it is easy to see from (5.39) that the next persymmetric relation holds

$$R'_{i,j} = R''_{p-i,p-j}. \quad (5.40)$$

If the next two vectors are introduced

$$\begin{aligned} \mathbf{y}_0^T &= (y_p, \dots, y_0) \\ \mathbf{y}_{N-1}^T &= (y_{N-1-p}, \dots, y_{N-1}), \end{aligned} \quad (5.41)$$

then the relations (5.39) can be written in vector notation

$$\begin{aligned} \mathbf{R}'_p &= \mathbf{R}_p - \mathbf{y}_0 \mathbf{y}_0^T - \mathbf{y}_{N-1} \mathbf{y}_{N-1}^T \\ \mathbf{R}''_p &= \mathbf{R}_p - \mathbf{y}_0^r \mathbf{y}_0^{rT} - \mathbf{y}_{N-1}^r \mathbf{y}_{N-1}^{rT}. \end{aligned} \quad (5.42)$$

The $(p+1)^{th}$ order matrix \mathbf{R}_{p+1} can be obtained from \mathbf{R}'_p or from \mathbf{R}''_p as follows

$$\mathbf{R}_{p+1} = \begin{bmatrix} & & & R_{0,p+1} \\ & \mathbf{R}'_p & & \vdots \\ & & & R_{p,p+1} \\ R_{p+1,0} & \dots & R_{p+1,p} & R_{p+1,p+1} \end{bmatrix} \quad (5.43)$$

$$\mathbf{R}_{p+1} = \begin{bmatrix} R_{0,0} & R_{0,1} & \dots & R_{0,p+1} \\ R_{1,0} & & & \\ \vdots & & \mathbf{R}''_p & \\ R_{p+1,0} & & & \end{bmatrix}. \quad (5.44)$$

Four auxiliary column vectors, $\mathbf{c}_p, \mathbf{c}''_p, \mathbf{d}_p, \mathbf{d}''_p$, are defined by the next matrix-vector products

$$\mathbf{R}_p \mathbf{c}_p = \mathbf{y}_0 \quad (5.45)$$

$$\mathbf{R}''_p \mathbf{c}''_p = \mathbf{y}_0 \quad (5.46)$$

$$\mathbf{R}_p \mathbf{d}_p = \mathbf{y}_{N-1} \quad (5.47)$$

$$\mathbf{R}''_p \mathbf{d}''_p = \mathbf{y}_{N-1}, \quad (5.48)$$

while for the reversed vectors (for example the reversed vector \mathbf{a}_p^r of \mathbf{a}_p is equal to $(a_{p,p}, \dots, a_{p,0})^T$) the next relations hold as a result of the persymmetry of the matrix \mathbf{R}_p

$$\mathbf{R}_p \mathbf{a}_p^r = \mathbf{e}_p^r \quad (5.49)$$

$$\mathbf{R}_p \mathbf{c}_p^r = \mathbf{y}_0^r \quad (5.50)$$

$$\mathbf{R}_p \mathbf{d}_p^r = \mathbf{y}_{N-1}^r. \quad (5.51)$$

From the identity $\mathbf{a}_p^T \mathbf{R}_p \mathbf{c}_p = \mathbf{c}_p^T \mathbf{R}_p^T \mathbf{a}_p$, from (5.34) and from (5.45) we obtain

$$\mathbf{a}_p^T \mathbf{y}_0 = \mathbf{c}_p^T \mathbf{e}_p, \quad (5.52)$$

which can be further reduced to

$$c_{p,0} = \frac{e_p(p)}{E_p}, \quad (5.53)$$

where $e_p(p)$ is obtained from (5.17) and can be written as

$$e_p(p) = \mathbf{y}_0^T \mathbf{a}_p. \quad (5.54)$$

If $r_p(N-1)$ from (5.17) is written as

$$r_p(N-1) = \mathbf{y}_{N-1}^T \mathbf{a}_p \quad (5.55)$$

we obtain from the identity $\mathbf{a}_p^T \mathbf{R}_p \mathbf{d}_p = \mathbf{d}_p^T \mathbf{R}_p^T \mathbf{a}_p$ and from (5.47) the value of $d_{p,0}$

$$d_{p,0} = \frac{r_p(N-1)}{E_p}. \quad (5.56)$$

From the identity $\mathbf{d}_p^T \mathbf{R}_p \mathbf{c}_p = \mathbf{c}_p^T \mathbf{R}_p^T \mathbf{d}_p$ and from (5.45) and (5.47) we obtain

$$h_p = \mathbf{y}_{N-1}^T \mathbf{c}_p = \mathbf{y}_0^T \mathbf{d}_p, \quad (5.57)$$

and from $\mathbf{d}_p^T \mathbf{R}_p \mathbf{c}_p^r = \mathbf{c}_p^{rT} \mathbf{R}_p^T \mathbf{d}_p$, (5.49) and (5.51) we find

$$s_p = \mathbf{y}_{N-1}^T \mathbf{c}_p = \mathbf{y}_0^T \mathbf{d}_p. \quad (5.58)$$

Two other scalars are introduced now

$$g_p = \mathbf{y}_0^T \mathbf{c}_p \quad (5.59)$$

$$w_p = \mathbf{y}_{N-1}^T \mathbf{d}_p. \quad (5.60)$$

If the time shift update of \mathbf{a}_p'

$$\mathbf{a}_p' = \alpha_p(\mathbf{a}_p + \beta_1 \mathbf{c}_p + \gamma_1 \mathbf{d}_p) \quad (5.61)$$

is substituted into (5.36) and if the relation for \mathbf{R}_p' of (5.42) is used the next relation is obtained

$$\mathbf{e}_p' = \alpha_p[\mathbf{e}_p + \{-e_p(p) + (1-g_p)\beta_1 - \gamma_1 h_p\}\mathbf{y}_0 + \{-r_p(N-1) - \beta_1 h_p + (1-w_p)\gamma_1\}\mathbf{y}_{N-1}], \quad (5.62)$$

but the next equality must hold

$$\mathbf{e}_p' = \alpha_p \mathbf{e}_p \quad (5.63)$$

which implies

$$\begin{aligned}(1 - g_p)\beta_1 - \gamma_1 h_p - e_p(p) &= 0 \\ -\beta_1 h_p + (1 - w_p)\gamma_1 - r_p(N - 1) &= 0.\end{aligned}$$

Solving these set of linear equations yields

$$\begin{aligned}\beta_1 &= [h_p r_p(N - 1) + e_p(p)(1 - w_p)]/D_p \\ \gamma_1 &= [r_p(N - 1)(1 - g_p) + h_p e_p(p)]/D_p,\end{aligned}\tag{5.64}$$

where the denominator D_p is given by

$$D_p = (1 - w_p)(1 - g_p) - h_p^2.\tag{5.65}$$

Because the first element of both \mathbf{a}_p and \mathbf{a}'_p are equal to one, α_p can be determined from

$$\alpha_p(1 + \beta_1 c_{p,0} + \gamma_1 d_{p,0}) = 1\tag{5.66}$$

by using (5.53), (5.56) and (5.64), resulting in

$$\alpha_p = \left[1 + \frac{e_p^2(p)(1 - w_p) + r_p^2(N - 1)(1 - g_p) + 2h_p e_p(p)r_p(N - 1)}{E_p D_p}\right]^{-1}.\tag{5.67}$$

The time shift update of \mathbf{c}''_p and of \mathbf{d}''_p is

$$\mathbf{c}''_p = \mathbf{c}_p + \beta_2 \mathbf{c}_p^r + \gamma_2 \mathbf{d}_p^r\tag{5.68}$$

$$\mathbf{d}''_p = \mathbf{d}_p + \beta_3 \mathbf{c}_p^r + \gamma_3 \mathbf{d}_p^r.\tag{5.69}$$

Premultiplying these both expressions by \mathbf{R}_p'' and the use of (5.46), (5.48) and the relation for \mathbf{R}_p'' of (5.42) gives

$$\begin{aligned}\mathbf{y}_0 &= \mathbf{y}_0 + \{-v_p + \beta_2(1 - g_p) - \gamma_2 h_p\}\mathbf{y}_0^r + \{-s_p - \beta_2 h_p + \gamma_2(1 - w_p)\}\mathbf{y}_{N-1}^r \\ \mathbf{y}_{N-1} &= \mathbf{y}_{N-1} + \{-s_p + \beta_3(1 - g_p) - \gamma_3 h_p\}\mathbf{y}_0^r + \{-u_p - \beta_3 h_p + \gamma_3(1 - w_p)\}\mathbf{y}_{N-1}^r,\end{aligned}\tag{5.70}$$

where the next two relations are also used

$$\begin{aligned}v_p &= \mathbf{y}_0^{r^T} \mathbf{c}_p \\ u_p &= \mathbf{y}_{N-1}^{r^T} \mathbf{d}_p.\end{aligned}\tag{5.71}$$

Because the last two terms of each expression of (5.71) must be zero, we obtain the values of the beta's and the gamma's as given in (5.72)

$$\begin{aligned}\beta_2 &= \frac{s_p h_p + v_p(1 - w_p)}{D_p} \\ \beta_3 &= \frac{u_p h_p + s_p(1 - w_p)}{D_p} \\ \gamma_2 &= \frac{v_p h_p + s_p(1 - g_p)}{D_p} \\ \gamma_3 &= \frac{s_p h_p + u_p(1 - g_p)}{D_p}.\end{aligned}\tag{5.72}$$

For the iteration to the $(p+1)^{th}$ model the next vectors are introduced (see also Appendix B)

$$\mathbf{a}_{p+1}^{1^T} = (\mathbf{a}_p'^T, 0) \quad (5.73)$$

$$\mathbf{e}_{p+1}^{1^T} = (\mathbf{e}_p'^T, \epsilon_p) \quad (5.74)$$

such that the following equation holds

$$\mathbf{R}_{p+1} \mathbf{a}_{p+1}^1 = \mathbf{e}_{p+1}^1. \quad (5.75)$$

This can be verified by the substitution of (5.43) into (5.75), the use of relation (5.36) and by making ϵ_p the inner product of the lowest row vector of \mathbf{R}_{p+1} and \mathbf{a}_{p+1}^1

$$\epsilon_p = \sum_{i=0}^p a'_{p,i} R_{p+1,i}. \quad (5.76)$$

The relation

$$\mathbf{R}_{p+1} \mathbf{a}_{p+1}^{1^r} = \mathbf{e}_{p+1}^{1^r} \quad (5.77)$$

can be checked by using (5.44) for \mathbf{R}_{p+1} , the persymmetric property (5.40) and the relation (5.36). To solve the equation

$$\mathbf{R}_{p+1} \mathbf{a}_{p+1} = \mathbf{e}_{p+1} \quad (5.78)$$

the vector

$$\mathbf{a}_{p+1} = \mathbf{a}_{p+1}^1 - \gamma_{p+1} \mathbf{a}_{p+1}^{1^r} \quad (5.79)$$

is proposed, giving us with (5.75) and (5.77)

$$\mathbf{e}_{p+1} = \mathbf{e}_{p+1}^1 - \gamma_{p+1} \mathbf{e}_{p+1}^{1^r}. \quad (5.80)$$

The first term of \mathbf{e}_{p+1} is equal to \mathbf{E}_{p+1} , while the other elements of \mathbf{e}_{p+1} are equal to zero, which gives us

$$\mathbf{E}_{p+1} = \mathbf{E}'_p - \gamma_{p+1} \epsilon_p \quad (5.81)$$

$$0 = \epsilon_p - \gamma_{p+1} \mathbf{E}'_p. \quad (5.82)$$

A combination of (5.81) and (5.82) yields

$$\gamma_{p+1} = \frac{\epsilon_p}{\mathbf{E}'_p} \quad (5.83)$$

$$\mathbf{E}_{p+1} = (1 - \gamma_{p+1}^2) \mathbf{E}'_p \quad (5.84)$$

and the recursion for $a_{p+1,i}$ is found from (5.79) and becomes

$$\begin{aligned} a_{p+1,i} &= a'_{p,i} - \gamma_{p+1} a'_{p,p+1-i}, 1 \leq i \leq p \\ a_{p+1,p+1} &= -\gamma_{p+1}. \end{aligned} \quad (5.85)$$

A comparison of these recursion with the Levinson-Durbin recursion of (5.18) shows that the new recursion is a function of the time-shifted parameter $a'_{p,i}$, rather than a function of $a_{p,i}$.

To increase the speed of the calculation of γ_{p+1} , we need a recursive relation for $R_{p+1,j}$ for the determination of ϵ_p . From (5.31) we can compute

$$\begin{aligned} R_{p,j-1} &= \sum_{n=p}^{N-1} y_{n-p} y_{n-j+1} + y_n y_{n-p+j-1} = \sum_{n=p}^{N-1} \{T_1^p(n) + T_2^p(n)\} \\ R_{p+1,j} &= \sum_{n=p+1}^{N-1} y_{n-p-1} y_{n-j} + y_n y_{n-p+j-1} = \sum_{n=p+1}^{N-1} \{T_1^{p+1}(n) + T_2^{p+1}(n)\} \end{aligned} \quad (5.86)$$

For $p+1 \leq n \leq N-1$ we see that $T_1^{p+1}(n) = T_1^p(n-1)$ and that $T_2^{p+1}(n) = T_2^p(n)$ so

$$R_{p+1,j} = \sum_{n=p+1}^{N-1} \{T_1^p(n-1) + T_2^p(n)\} = \sum_{n=p}^{N-1} \{T_1^p(n) + T_2^p(n)\} - T_1^p(N-1) - T_2^p(p) \quad (5.87)$$

or

$$R_{p+1,j} = R_{p,j-1} - y_p y_{j-1} - y_{N-1-p} y_{N-j} \quad (5.88)$$

for $1 \leq j \leq p+1$.

The order update relationships for \mathbf{c}_{p+1} and \mathbf{d}_{p+1} are

$$\mathbf{c}_{p+1} = \begin{pmatrix} 0 \\ \mathbf{c}_p'' \end{pmatrix} + \alpha_2 \mathbf{a}_{p+1} \quad (5.89)$$

$$\mathbf{d}_{p+1} = \begin{pmatrix} 0 \\ \mathbf{d}_p'' \end{pmatrix} + \alpha_3 \mathbf{a}_{p+1}. \quad (5.90)$$

Since the first element of \mathbf{a}_{p+1} is one, we have with (5.53) and with (5.56)

$$\begin{aligned} \alpha_2 &= c_{p+1,0} = \frac{e_{p+1}(p+1)}{E_{p+1}} \\ \alpha_3 &= d_{p+1,0} = \frac{r_{p+1}(N-1)}{E_{p+1}}. \end{aligned} \quad (5.91)$$

From the definition of g_p in (5.59) and using (5.68) and (5.89) we get

$$\begin{aligned} g_{p+1} &= (y_{p+1}, \dots, y_0) \mathbf{c}_{p+1} \\ &= (y_{p+1}, \dots, y_0) \left(\begin{pmatrix} 0 \\ \mathbf{c}_p + \beta_2 \mathbf{c}_p^r + \gamma_2 \mathbf{d}_p^r \end{pmatrix} + \alpha_2 \mathbf{a}_{p+1} \right), \end{aligned}$$

which gives us

$$g_{p+1} = g_p + \frac{e_{p+1}^2(p+1)}{E_{p+1}} + \frac{v_p^2(1-w_p) + s_p^2(1-g_p) + 2s_p h_p v_p}{D_p}. \quad (5.92)$$

From the definition of w_p in (5.60) and using (5.69) and (5.90) we find in the same way

$$w_{p+1} = w_p + \frac{r_{p+1}^2(N-1)}{E_{p+1}} + \frac{s_p^2(1-w_p) + u_p^2(1-g_p) + 2s_p h_p u_p}{D_p}. \quad (5.93)$$

The initializing values for $p = 0$ are given now. From (5.17) we see that $e_0(n) = r_0(n) = y_n$ for $0 \leq n \leq N - 1$, so from (5.20) and from (5.17) we see

$$\begin{aligned} E_0 &= 2 \sum_{n=0}^{N-1} y_n^2 \\ e_0(0) &= y_0 \\ r_0(N-1) &= y_{N-1}. \end{aligned}$$

From (5.53) and (5.56) we obtain

$$\begin{aligned} c_{0,0} &= \frac{y_0}{E_0} \\ d_{0,0} &= \frac{y_{N-1}}{E_0}, \end{aligned}$$

and from (5.92) and (5.93), or from (5.59) and (5.60)

$$\begin{aligned} g_0 &= \frac{y_0^2}{E_0} \\ w_0 &= \frac{y_{N-1}^2}{E_0}. \end{aligned}$$

From (5.57), (5.58) and (5.71) we find the scalars

$$\begin{aligned} h_0 &= \frac{y_0 y_{N-1}}{E_0} \\ s_0 &= \frac{y_0 y_{N-1}}{E_0} \\ v_0 &= \frac{y_0^2}{E_0} \\ u_0 &= \frac{y_{N-1}^2}{E_0}. \end{aligned}$$

From (5.65) we obtain D_0 and because α_0 from (5.67) is equal to D_0 we find E'_0 from (5.63)

$$\begin{aligned} D_0 &= 1 - g_0 - w_0 \\ E'_0 &= E_0 D_0 = E_0 - y_0^2 - y_{N-1}^2. \end{aligned}$$

From (5.68) and (5.69) we obtain with (5.72)

$$\begin{aligned} c''_{0,0} &= \frac{y_0}{E'_0} \\ d''_{0,0} &= \frac{y_{N-1}}{E'_0}. \end{aligned}$$

For $p = 1$ we find from (5.31)

$$R_{1,0} = 2 \sum_{n=1}^{N-1} y_n y_{n-1},$$

from (5.85), (5.83) and (5.76)

$$a_{1,1} = -\gamma_1 = -\frac{R_{1,0}}{E'_0}$$

and from (5.84)

$$E_1 = (1 - \gamma_1^2)E'_0.$$

The *Marple algorithm* becomes now

step1 Initialize at $p = 0$,

step2 Calculate from (5.54) and (5.55)

$$\begin{aligned} e_p(p) &= y_p + \sum_{i=1}^p a_{p,i} y_{p-i} \\ r_p(N-1) &= y_{N-1-p} + \sum_{i=1}^p a_{p,i} y_{N-1-p+i} \end{aligned}$$

step3 Calculate $c_{p,0} = \alpha_2$ from (5.53) and $d_{p,0} = \alpha_3$ from (5.56) and from (5.89) and (5.90) :

$$\begin{aligned} c_{p,i} &= c''_{p-1,i-1} + \alpha_2 a_{p,i} \\ d_{p,i} &= d''_{p-1,i-1} + \alpha_3 a_{p,i} \end{aligned}$$

step4 Calculate according to (5.92) and (5.93)

$$\begin{aligned} g_p &= g_{p-1} + \frac{e_p^2(p)}{E_p} + \frac{v_{p-1}^2(1 - w_{p-1}) + s_{p-1}^2(1 - g_{p-1}) + 2s_{p-1}h_{p-1}v_{p-1}}{D_{p-1}} \\ w_p &= w_{p-1} + \frac{r_p^2(N-1)}{E_p} + \frac{s_{p-1}^2(1 - w_{p-1}) + u_{p-1}^2(1 - g_{p-1}) + 2s_{p-1}h_{p-1}u_{p-1}}{D_{p-1}}. \end{aligned}$$

and following (5.57), (5.58) and (5.71)

$$\begin{aligned} h_p &= \sum_{i=0}^p y_{N-1-p+i} c_{p,i} \\ s_p &= \sum_{i=0}^p y_{N-1-i} c_{p,i} \\ v_p &= \sum_{i=0}^p y_i c_{p,i} \\ u_p &= \sum_{i=0}^p y_{N-1-i} d_{p,i} \end{aligned}$$

step5 Calculate the denominator from (5.65)

$$D_p = (1 - w_p)(1 - g_p) - h_p^2$$

- step6** Calculate α_p with (5.67), from (5.63) $E'_p = \alpha_p E_p$, from (5.61) and (5.64) a'_p , and from (5.68), (5.69) and (5.72) c''_p and d''_p
- step7** Calculate $R_{p+1,j}$ for $1 \leq j \leq p$ from (5.88), $R_{p+1,0}$ from (5.31) and ϵ_p with (5.76). Then γ_{p+1} and E_{p+1} are determined with (5.83) and (5.84), next the predictor coefficients are updated according to (5.85)
- step8** $p := p + 1$ and goto step2 until $p > M$

5.5 The Morf algorithm

In the Burg and the Marple algorithm the equations (5.17) were used for the prediction errors. It was assumed that the optimizing prediction coefficients were the same for the forward and for the reverse prediction. Now the two predictors are separated by the use of different coefficients [17].

$$\begin{aligned} e_p(n) &= \sum_{i=0}^p a_{p,i} y_{n-i} \\ r_p(n) &= \sum_{i=0}^p b_{p,i} y_{n-p+i}. \end{aligned} \quad (5.94)$$

In the two previous algorithms the same expression (5.20) for the energy to be minimized was used. Now two separated energy terms for the forward and for the reverse predictor are used.

$$\begin{aligned} E_p &= \sum_{n=p}^{N-1} e_p^2(n) \\ R_p &= \sum_{n=p}^{N-1} r_p^2(n) \end{aligned} \quad (5.95)$$

Minimizing E_p with respect to the a_i 's gives

$$\mathbf{R}_p \mathbf{a}_p = \mathbf{R}_p (1, a_{p,1}, \dots, a_{p,p})^T = \mathbf{e}_p = (E_p, 0, \dots, 0)^T, \quad (5.96)$$

where the entries of \mathbf{R}_p are $R_{i,j} = \sum_{n=p}^{N-1} y_{n-i} y_{n-j}$. Note that the matrix \mathbf{R}_p is equal to the earlier mentioned covariance matrix. Minimizing R_p with respect to the b_i 's gives

$$\mathbf{R}_p \mathbf{b}_p = \mathbf{R}_p (b_{p,p}, \dots, b_{p,1}, 1)^T = \mathbf{r}_p = (0, \dots, 0, R_p)^T. \quad (5.97)$$

Two additional energy terms are introduced

$$\begin{aligned} E'_p &= \sum_{n=p}^{N-2} e_p^2(n+1) \\ R'_p &= \sum_{n=p+1}^{N-1} r_p^2(n-1), \end{aligned} \quad (5.98)$$

and note that the range for $e_p(n+1)$ is decreased from $p-1 \leq n \leq N-2$ to $p \leq n \leq N-2$ and that the range for $r_p(n-1)$ is also slightly modified. Minimizing both energy terms yields

$$\mathbf{R}'_p \mathbf{a}'_p = \mathbf{e}'_p \quad (5.99)$$

$$\mathbf{R}''_p \mathbf{b}'_p = \mathbf{r}'_p. \quad (5.100)$$

The entries of the matrices \mathbf{R}'_p and \mathbf{R}''_p are

$$\begin{aligned} R'_{i,j} &= \sum_{n=p}^{N-2} y_{n+1-i} y_{n+1-j} \\ R''_{i,j} &= \sum_{n=p+1}^{N-2} y_{n-1-i} y_{n-1-j} \end{aligned} \quad (5.101)$$

for $0 \leq i, j \leq p$ and are related with $R_{i,j}$ as follows

$$\begin{aligned} R'_{i,j} &= R_{i,j} - y_{p-i} y_{p-j} \\ R''_{i,j} &= R_{i,j} - y_{N-1-i} y_{N-1-j}. \end{aligned} \quad (5.102)$$

If the vectors of (5.41) are used, these relations changes into

$$\begin{aligned} \mathbf{R}'_p &= \mathbf{R}_p - \mathbf{y}_0 \mathbf{y}_0^T \\ \mathbf{R}''_p &= \mathbf{R}_p - \mathbf{y}_{N-1}^r \mathbf{y}_{N-1}^{rT}. \end{aligned} \quad (5.103)$$

The vector \mathbf{y}_{N-1}^r has the same elements as the vector \mathbf{y}_{N-1} but with the order reversed. As in analogy with the Marple algorithm, we introduce four column vectors $\mathbf{c}_p, \mathbf{c}'_p, \mathbf{d}_p$ and \mathbf{d}'_p in the following way

$$\mathbf{R}_p \mathbf{c}_p = \mathbf{y}_0 \quad (5.104)$$

$$\mathbf{R}''_p \mathbf{c}'_p = \mathbf{y}_0 \quad (5.105)$$

$$\mathbf{R}_p \mathbf{d}_p = \mathbf{y}_{N-1}^r \quad (5.106)$$

$$\mathbf{R}'_p \mathbf{d}'_p = \mathbf{y}_{N-1}^r, \quad (5.107)$$

and in the same manner as in the previous section we can obtaine the next relations

$$c_{p,0} = \frac{e_p(p)}{E_p} \quad (5.108)$$

$$d_{p,p} = \frac{r_p(N-1)}{R_p} \quad (5.109)$$

with

$$e_p(p) = \mathbf{y}_0^T \mathbf{a}_p \quad (5.110)$$

$$r_p(N-1) = \mathbf{y}_{N-1}^{rT} \mathbf{b}_p \quad (5.111)$$

$$h_p = \mathbf{y}_{N-1}^{rT} \mathbf{c}_p = \mathbf{y}_0^T \mathbf{d}_p. \quad (5.112)$$

If the time shift update of \mathbf{a}'_p and \mathbf{b}'_p

$$\begin{aligned} \mathbf{a}'_p &= (1, a'_{p,1}, \dots, a'_{p,p})^T = \alpha'_p (\mathbf{a}_p + \beta_1 \mathbf{c}_p) \\ \mathbf{b}'_p &= (b'_{p,p}, \dots, b'_{p,1}, 1)^T = \alpha''_p (\mathbf{b}_p + \beta_2 \mathbf{d}_p) \end{aligned} \quad (5.113)$$

is inserted into (5.99) and (5.100) and the relation (5.103) is used, we obtain

$$\begin{aligned} \mathbf{e}'_p &= \alpha'_p[\mathbf{e}_p + \{-e_p(p) + (1 - g_p)\beta_1\}\mathbf{y}_0] \\ \mathbf{r}'_p &= \alpha''_p[\mathbf{r}_p + \{-r_p(N - 1) + (1 - w_p)\beta_2\}\mathbf{y}_{N-1}^T], \end{aligned} \quad (5.114)$$

with g_p and w_p defined as

$$\begin{aligned} g_p &= \mathbf{y}_0^T \mathbf{c}_p \\ w_p &= \mathbf{y}_{N-1}^T \mathbf{d}_p. \end{aligned} \quad (5.115)$$

Because the last terms of (5.114) must be zero, we find

$$\begin{aligned} \beta_1 &= \frac{e_p(p)}{1 - g_p} \\ \beta_2 &= \frac{r_p(N - 1)}{1 - w_p}, \end{aligned} \quad (5.116)$$

and because the first term of \mathbf{a}_p and \mathbf{a}'_p and the last term of \mathbf{b}_p and \mathbf{b}'_p are one we have

$$\begin{aligned} \alpha'_p &= [1 + \frac{e_p^2(p)}{E_p(1 - g_p)}]^{-1} \\ \alpha''_p &= [1 + \frac{r_p^2(N - 1)}{R_p(1 - w_p)}]^{-1}. \end{aligned} \quad (5.117)$$

Using these values of the α 's and β 's in (5.113) and in (5.114) we find the next recursions

$$\begin{aligned} \mathbf{a}'_p &= [\mathbf{a}_p + \frac{e_p(p)}{1 - g_p} \mathbf{c}_p][1 + \frac{e_p^2(p)}{E_p(1 - g_p)}]^{-1} \\ \mathbf{b}'_p &= [\mathbf{b}_p + \frac{r_p(N - 1)}{1 - w_p} \mathbf{d}_p][1 + \frac{r_p^2(N - 1)}{R_p(1 - w_p)}]^{-1} \\ \mathbf{e}'_p &= [1 + \frac{e_p^2(p)}{E_p(1 - g_p)}]^{-1} \mathbf{e}_p \\ \mathbf{r}'_p &= [1 + \frac{r_p^2(N - 1)}{R_p(1 - w_p)}]^{-1} \mathbf{r}_p. \end{aligned} \quad (5.118)$$

The time shift update of \mathbf{c}'_p and of \mathbf{d}'_p is

$$\begin{aligned} \mathbf{c}'_p &= \mathbf{c}_p + \beta_3 \mathbf{d}_p \\ \mathbf{d}'_p &= \mathbf{d}_p + \beta_4 \mathbf{c}_p. \end{aligned} \quad (5.119)$$

Premultiplying the first expression by \mathbf{R}''_p and the second term by \mathbf{R}'_p , the use of (5.105) and of (5.107) and the relationships of \mathbf{R}''_p and of \mathbf{R}'_p given in (5.103) gives us

$$\begin{aligned} \mathbf{y}_0 &= \mathbf{y}_0 + \{-h_p + \beta_3(1 - w_p)\}\mathbf{y}_{N-1}^T \\ \mathbf{y}_{N-1}^T &= \{-h_p + \beta_4(1 - g_p)\}\mathbf{y}_0 + \mathbf{y}_{N-1}^T. \end{aligned} \quad (5.120)$$

These equations gives us the values of the β 's and the substitution into (5.119) yields the next recursions

$$\begin{aligned} \mathbf{c}'_p &= \mathbf{c}_p + \frac{h_p}{1 - w_p} \mathbf{d}_p \\ \mathbf{d}'_p &= \mathbf{d}_p + \frac{h_p}{1 - g_p} \mathbf{c}_p. \end{aligned} \quad (5.121)$$

For the iteration to the $(p + 1)^{th}$ model the next vectors are introduced

$$\mathbf{a}_{p+1}^{1^T} = (\mathbf{a}'_p, 0) \quad (5.122)$$

$$\mathbf{b}_{p+1}^{1^T} = (0, \mathbf{b}'_p) \quad (5.123)$$

$$\mathbf{e}_{p+1}^{1^T} = (\mathbf{e}'_p, \epsilon_p^+) \quad (5.124)$$

$$\mathbf{r}_{p+1}^{1^T} = (\epsilon_p^-, \mathbf{r}'_p) \quad (5.125)$$

such that the following equation holds

$$\mathbf{R}_{p+1} \mathbf{a}_{p+1}^1 = \mathbf{e}_{p+1}^1 \quad (5.126)$$

$$\mathbf{R}_{p+1} \mathbf{b}_{p+1}^1 = \mathbf{r}_{p+1}^1 \quad (5.127)$$

and with

$$\begin{aligned} \epsilon_p^+ &= \sum_{i=0}^p a'_{p,i} R_{p+1,i} \\ \epsilon_p^- &= \sum_{i=0}^p b'_{p,i} R_{0,p+1-i}. \end{aligned} \quad (5.128)$$

Analogue to the Marple algorithm the recursion relations for the predictor coefficients $a_{p+1,i}$ and $b_{p+1,i}$ are

$$\begin{aligned} a_{p+1,i} &= a'_{p,i} - \gamma_{p+1} b'_{p,p+1-i}, \quad 1 \leq i \leq p \\ a_{p+1,p+1} &= -\gamma_{p+1} \\ b_{p+1,i} &= b'_{p,i} - \beta_{p+1} a'_{p,p+1-i}, \quad 1 \leq i \leq p \\ b_{p+1,p+1} &= -\beta_{p+1}, \end{aligned} \quad (5.129)$$

with

$$\gamma_{p+1} = \frac{\epsilon_p^+}{R'_p} \quad (5.130)$$

$$\beta_{p+1} = \frac{\epsilon_p^-}{E'_p} \quad (5.131)$$

$$E_{p+1} = E'_p - \gamma_{p+1} \epsilon_p^- \quad (5.132)$$

$$R_{p+1} = R'_p - \beta_{p+1} \epsilon_p^+. \quad (5.133)$$

The order update relationships for \mathbf{c}_{p+1} and \mathbf{d}_{p+1} are

$$\mathbf{c}_{p+1} = \begin{pmatrix} 0 \\ \mathbf{c}'_p \end{pmatrix} + \alpha_2 \mathbf{a}_{p+1} \quad (5.134)$$

$$\mathbf{d}_{p+1} = \begin{pmatrix} \mathbf{d}'_p \\ 0 \end{pmatrix} + \alpha_3 \mathbf{b}_{p+1}. \quad (5.135)$$

Since the first element of \mathbf{a}_{p+1} and the last term of \mathbf{b}_{p+1} is one, we have with (5.108) and with (5.109)

$$\begin{aligned} \alpha_2 &= c_{p+1,0} = \frac{e_{p+1}(p+1)}{E_{p+1}} \\ \alpha_3 &= d_{p+1,p+1} = \frac{r_{p+1}(N-1)}{R_{p+1}}. \end{aligned} \quad (5.136)$$

From the definition of g_p in (5.115) and using (5.121) and (5.134) we get

$$\begin{aligned} g_{p+1} &= (y_{p+1}, \dots, y_0) \mathbf{c}_{p+1} \\ &= (y_{p+1}, \dots, y_0) \left(\begin{pmatrix} 0 \\ \mathbf{c}_p + \frac{h_p}{1-w_p} \mathbf{d}_p \end{pmatrix} + \alpha_2 \mathbf{a}_{p+1} \right), \end{aligned}$$

which gives us

$$g_{p+1} = g_p + \frac{e_{p+1}^2(p+1)}{E_{p+1}} + \frac{h_p^2}{1-w_p}. \quad (5.137)$$

From the definition of w_p in (5.115) and using (5.121) and (5.135) we find in the same way

$$w_{p+1} = w_p + \frac{r_{p+1}^2(N-1)}{R_{p+1}} + \frac{h_p^2}{1-g_p}. \quad (5.138)$$

In the same manner as in the previous section (see (5.86) and (5.87)), we can prove that the next recursions exists

$$\begin{aligned} \mathbf{R}_{p+1,j} &= \mathbf{R}_{p,j-1} - y_{N-1-p} y_{N-j} \\ \mathbf{R}_{0,p+1-j} &= \mathbf{R}_{0,p-(j-1)} - y_p y_{j-1}, \end{aligned} \quad (5.139)$$

which can be used to calculate the γ 's and β 's more effective.

The *Morf algorithm* becomes now

step1 Initialize for $p = 0$.

step2 Determine $e_p(p)$ from (5.110) and $r_p(N-1)$ from (5.111).

step3 Calculate \mathbf{c}_p and \mathbf{d}_p with (5.108), (5.109), (5.136), (5.134) and (5.135).

step4 Obtain g_p and w_p from (5.137) and (5.138), and h_p from (5.112).

step5 Determine $\mathbf{a}'_p, \mathbf{b}'_p, \mathbf{e}'_p$ and \mathbf{r}'_p from (5.118), and \mathbf{c}'_p and \mathbf{d}'_p with (5.121).

step6 Calculate $E'_p = \alpha'_p E_p$ and $R'_p = \alpha''_p R_p$ with (5.117).

step7 Calculate ϵ_p^+ and ϵ_p^- with (5.128), γ_{p+1} and β_{p+1} with (5.130) and (5.131), E_{p+1} and R_{p+1} with (5.132) and (5.133) and the prediction coefficients with (5.129).

step8 $p := p + 1$ and goto step2 until $p > M$.

Appendix A

Mathematical preliminaries

A.1 Review of linear spaces and inner products

Ordinary Euclidian space is the most familiar example of a linear space or vector space. In Euclidian space, a vector is a point in the space, and is specified by its coordinates, n coordinates in an n -dimensional space. The notation

$$\mathbf{X} \leftrightarrow (x_1, x_2, \dots, x_n)$$

means that the vector \mathbf{X} corresponds with the components x_1, \dots, x_n . There are rules for adding two vectors (sum the individual components) and multiplying a vector by a scalar (multiply each component by that scalar). The linear space concept can be generalized in the following fashion. Formally a linear space is a set H of elements or vectors of the set, together with a rule for adding two vectors in the space to generate another vector and a rule for multiplying a vector by a scalar to generate another vector. A vector in the space will be denoted by a bold-faced letter. The addition rule associates with the sum of two vectors $\mathbf{X} + \mathbf{Y}$ another vector, and must obey the ordinary rules of arithmetic, including commutative and associative laws,

$$\begin{aligned}\mathbf{X} + \mathbf{Y} &= \mathbf{Y} + \mathbf{X} \\ \mathbf{X} + (\mathbf{Y} + \mathbf{Z}) &= (\mathbf{X} + \mathbf{Y}) + \mathbf{Z}\end{aligned}$$

The linear space must include a zero vector $\mathbf{0}$, with the property that

$$\mathbf{0} + \mathbf{X} = \mathbf{X}$$

and there must for every vector \mathbf{X} be another vector $-\mathbf{X}$ with the property that

$$\mathbf{X} + (-\mathbf{X}) = \mathbf{0}.$$

The rule for multiplication by a scalar associates with scalar α and vector \mathbf{X} another vector $\alpha.\mathbf{X}$ which must obey the associative law,

$$\alpha.(\beta.\mathbf{X}) = (\alpha\beta).\mathbf{X}$$

and also follow the rules

$$1.\mathbf{X} = \mathbf{X}$$

and $0.X = 0$. Finally, addition and multiplication must obey the distributive laws,

$$\begin{aligned}\alpha.(X + Y) &= \alpha.X + \alpha.Y \\ (\alpha + \beta).X &= \alpha.X + \beta.X\end{aligned}$$

The definition of Euclidean space given earlier meets all these requirements and is therefore a linear space. Another linear space is the space of random variables with finite second moments. Let X be a random variable with zero mean and finite second moment,

$$\mathbb{E}[X] = 0$$

$$\mathbb{E}[X^2] < \infty. \quad (\text{A.1})$$

The collection of all such random variables can be considered as a linear space, where the vectors correspond to random variables,

$$X \leftrightarrow X.$$

To complete the definition of this space, 0 is defined as the random variable which is always zero,

$$0 \leftrightarrow 0,$$

and the vector $\alpha.X$ corresponds to the random variable αX . The sum of two vectors corresponds to the sum of the corresponding random variables,

$$X + Y \leftrightarrow X + Y.$$

The definition of linear space does not capture the most important properties of Euclidean space; namely, the geometric structure. This structure includes such concepts as the length of a vector in the space, and the angle between two vectors. All these properties of Euclidean space can be deduced from the definition of inner product two vectors. This inner product $\langle X, Y \rangle$ is a real-valued quantity defined for Euclidean space as

$$\langle X, Y \rangle = \sum_{i=1}^n x_i y_i.$$

A special notation

$$\|X\|^2 = \langle X, X \rangle = \sum_{i=1}^n x_i^2 \quad (\text{A.2})$$

can be introduced, where $\|X\|$ is called the norm of the vector X . It has the geometric interpretation as the length of the vector. The inner product of two vectors is equal to the product of the length of the first vector, the length of the second vector and the cosine of the angle between the vectors. A case of special interest is where the two vectors are perpendicular or orthogonal, in which case the inner product is zero.

The inner product as applied to Euclidean space can be generalized to other linear spaces of interest. The important consequence is that the geometric concepts familiar in Euclidean space can be applied to these spaces as well. Let X and Y be vectors of a linear space, and

suppose that an inner product $\langle \mathbf{X}, \mathbf{Y} \rangle$ of two vectors is defined on that space. This inner product is a scalar, and must obey the rules

$$\begin{aligned}\langle \mathbf{X} + \mathbf{Y}, \mathbf{Z} \rangle &= \langle \mathbf{X}, \mathbf{Z} \rangle + \langle \mathbf{Y}, \mathbf{Z} \rangle \\ \langle \alpha \mathbf{X}, \mathbf{Y} \rangle &= \alpha \langle \mathbf{X}, \mathbf{Y} \rangle \\ \langle \mathbf{X}, \mathbf{Y} \rangle &= \langle \mathbf{Y}, \mathbf{X} \rangle \\ \langle \mathbf{X}, \mathbf{X} \rangle &\geq 0, \mathbf{X} \neq 0.\end{aligned}$$

For the space of random variables with finite second moment the inner product can be defined as

$$\langle \mathbf{X}, \mathbf{Y} \rangle = E[XY].$$

The norm, as defined in (A.2) becomes

$$\|\mathbf{X}\|^2 = E[X^2],$$

and the condition of (A.1) corresponds to the assumption that the vector has finite norm or length.

The geometric properties are so important that the special name inner product space is given to a linear space on which an inner product is defined. If the inner product space has the additional property of completeness, it is defined to be a Hilbert space. Intuitively the notion of completeness means that there are not "missing" vectors that are arbitrarily close to vectors in the space but are not themselves in the space.

Another important object is the subspace of a linear space. This is a subset of the linear space which is itself a linear space. An example of a subspace is the set of vectors obtained by forming all possible weighted linear combinations of n vectors $\mathbf{X}_1, \dots, \mathbf{X}_n$. The subspace so formed is said to be spanned by the set of n vectors.

A.2 The projection theorem

Given a subspace M of a Hilbert space H and a vector \mathbf{X} in H there is an unique vector $P_M \mathbf{X}$ in M called the projection of \mathbf{X} on M which has the property that

$$\langle \mathbf{X} - P_M \mathbf{X}, \mathbf{Y} \rangle = 0$$

for every vector \mathbf{Y} in M . A consequence of the theorem is that the projection $P_M \mathbf{X}$ is the unique vector in M which is closest to \mathbf{X} ; That is

$$\|\mathbf{X} - P_M \mathbf{X}\| < \|\mathbf{X} - \mathbf{Y}\|$$

for every $\mathbf{Y} \neq P_M \mathbf{X}$ in M .

A.3 Orthogonality principle revisited

Equation (2.2) can be written as

$$e_n = y_n - \sum_{i=1}^p c_i y_{n-i} \quad (\text{A.3})$$

If the next vectors are introduced $\mathbf{E} \leftrightarrow e_n$, $\mathbf{Y} \leftrightarrow y_n$ and $\mathbf{Y}_i \leftrightarrow y_{n-i}$ for $1 \leq i \leq p$, equation (A.3) gives

$$\mathbf{E} = \mathbf{Y} - \mathbf{Y}_M.$$

The vector \mathbf{Y}_M is in a subspace M spanned by the p vectors \mathbf{Y}_i . To minimize (2.1) or to minimize $\|\mathbf{E}\|^2 = \|\mathbf{Y} - \mathbf{Y}_M\|^2$ the constants c_i are choosen such that $\mathbf{Y}_M = \mathbf{P}_M \mathbf{Y}$; \mathbf{Y}_M is the projection of \mathbf{Y} on M . Then \mathbf{E} is orthogonal to each vector in M and thus

$$\langle \mathbf{E}, \mathbf{Y}_i \rangle = 0, \quad 1 \leq i \leq p. \quad (\text{A.4})$$

The relation (A.4) corresponds to (2.3)

Appendix B

The Levinson-Durbin recursion in matrix form

B.1 The symmetric or Hermitian Toeplitz situation

In matrix form, the equation (2.6) for a p^{th} order model becomes

$$\mathbf{R}_p \mathbf{a}_p = \mathbf{e}_p, \quad (\text{B.1})$$

where \mathbf{R}_p is a $(p+1) \times (p+1)$ symmetric matrix with elements $R(|j-i|)$. The $(p+1) \times 1$ column vectors \mathbf{a}_p and \mathbf{e}_p are

$$\begin{aligned} \mathbf{a}_p^T &= (1, a_{p1}, a_{p2}, \dots, a_{pp}) \\ \mathbf{e}_p^T &= (E_p, 0, 0, \dots, 0) \end{aligned} \quad (\text{B.2})$$

The autocorrelation matrix \mathbf{R}_p has two properties where the iteration is based on :

- i. the matrix of given order contains as subblocks all the lower order matrices
- ii. the matrix is reflection invariant: it remains invariant under the interchange of its columns and then of its rows.

The last property implies that if, for certain vectors \mathbf{c}_p and \mathbf{d}_p ,

$$\mathbf{R}_p \mathbf{c}_p = \mathbf{d}_p, \quad (\text{B.3})$$

then

$$\mathbf{R}_p \mathbf{c}_p^r = \mathbf{d}_p^r, \quad (\text{B.4})$$

where \mathbf{c}_p^r and \mathbf{d}_p^r are just the vectors \mathbf{c}_p and \mathbf{d}_p in reverse order. If

$$\mathbf{c}_p^T = (c_0, c_1, \dots, c_{p-1}, c_p),$$

then

$$\mathbf{c}_p^{rT} = (c_p, c_{p-1}, \dots, c_1, c_0).$$

Assume that equation (B.1) is solved. For the iteration to the $(p + 1)^{th}$ model the next vectors are introduced

$$\begin{aligned} \mathbf{a}_{p+1}^{1T} &= (\mathbf{a}_p^T, 0) \\ \mathbf{e}_{p+1}^{1T} &= (\mathbf{e}_p^T, \epsilon_p) \end{aligned}$$

such that the following equation holds

$$\mathbf{R}_{p+1} \mathbf{a}_{p+1}^1 = \mathbf{e}_{p+1}^1. \quad (\text{B.5})$$

Then according to (B.4)

$$\mathbf{R}_{p+1} \mathbf{a}_{p+1}^{1r} = \mathbf{e}_{p+1}^{1r}. \quad (\text{B.6})$$

For the solution of the equation

$$\mathbf{R}_{p+1} \mathbf{a}_{p+1} = \mathbf{e}_{p+1} \quad (\text{B.7})$$

the vector

$$\mathbf{a}_{p+1} = \mathbf{a}_{p+1}^1 - \gamma_{p+1} \mathbf{a}_{p+1}^{1r} \quad (\text{B.8})$$

is proposed. Then, substitution of (B.8) in (B.7) and using the equations (B.5) and (B.6), we have

$$\mathbf{e}_{p+1} = \mathbf{e}_{p+1}^1 - \gamma_{p+1} \mathbf{e}_{p+1}^{1r}$$

or

$$\mathbf{E}_{p+1} = \mathbf{E}_p - \gamma_{p+1} \mathbf{E}_p \quad (\text{B.9})$$

and

$$\epsilon_p - \gamma_{p+1} \mathbf{E}_p = 0. \quad (\text{B.10})$$

Combinations of (B.9) and (B.10) give

$$\gamma_{p+1} = \frac{\epsilon_p}{\mathbf{E}_p} \quad (\text{B.11})$$

$$\mathbf{E}_{p+1} = (1 - \gamma_{p+1}^2) \mathbf{E}_p, \quad (\text{B.12})$$

where ϵ_p can be found from (B.5) and equals the inner product of the lowest row vector of \mathbf{R}_{p+1} and \mathbf{a}_{p+1}^1

$$\epsilon_p = \sum_{i=0}^p a_{p,i} R(p+1-i).$$

B.2 The non-symmetric Toeplitz situation

The previous result is found for a special case of the general situation, where the matrix \mathbf{R}_p is a non-symmetric Toeplitz matrix with elements $R(j-i)$. The matrix is Toeplitz with diagonal disagreement. So (2.6) generalizes to

$$\begin{bmatrix} R(0) & R(1) & R(p) \\ R(-1) & R(0) & R(p-1) \\ \vdots & \vdots & \vdots \\ R(-p) & R(-p+1) & R(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_{p1} \\ \vdots \\ a_{pp} \end{bmatrix} = \begin{bmatrix} \mathbf{E}_p \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (\text{B.13})$$

or

$$\mathbf{R}_p \mathbf{a}_p = \mathbf{e}_p, \quad (\text{B.14})$$

the equations for the forward prediction. Simular equations for the backward prediction can be introduced

$$\mathbf{R}_p \mathbf{b}_p = \mathbf{r}_p. \quad (\text{B.15})$$

The $(p+1) \times 1$ column vectors \mathbf{b}_p and \mathbf{r}_p are

$$\begin{aligned} \mathbf{b}_p^T &= (b_{pp}, \dots, b_{p1}, 1) \\ \mathbf{r}_p^T &= (0, \dots, 0, R_p) \end{aligned}$$

As in section B.1 the vectors \mathbf{a}_{p+1}^1 and \mathbf{e}_{p+1}^1 are introduced for the iteration of (B.14). For the iteration of (B.15) the next vectors are used

$$\begin{aligned} \mathbf{b}_{p+1}^{1T} &= (0, \mathbf{b}_p^T) \\ \mathbf{r}_{p+1}^{1T} &= (\epsilon_p^-, \mathbf{r}_p^T). \end{aligned}$$

So the following equations hold

$$\begin{aligned} \mathbf{R}_{p+1} \mathbf{a}_{p+1}^1 &= \mathbf{e}_{p+1}^1 \\ \mathbf{R}_{p+1} \mathbf{b}_{p+1}^1 &= \mathbf{r}_{p+1}^1, \end{aligned} \quad (\text{B.16})$$

where ϵ_p and ϵ_p^- are

$$\begin{aligned} \epsilon_p &= \sum_{i=0}^p a_{p,i} R(-p-1+i), \\ \epsilon_p^- &= \sum_{i=0}^p b_{p,i} R(p+1-i). \end{aligned} \quad (\text{B.17})$$

For (B.16) we use the next short notation

$$\mathbf{R}_{p+1} [\mathbf{a}_{p+1}^1 \mathbf{b}_{p+1}^1] = [\mathbf{e}_{p+1}^1 \mathbf{r}_{p+1}^1] \quad (\text{B.18})$$

to solve

$$\mathbf{R}_{p+1} [\mathbf{a}_{p+1} \mathbf{b}_{p+1}] = [\mathbf{e}_{p+1} \mathbf{r}_{p+1}]. \quad (\text{B.19})$$

Therefor a 2×2 matrix \mathbf{F} is introduced as

$$\mathbf{F} = \begin{bmatrix} 1 & -\epsilon_p^- \mathbf{E}_p^{-1} \\ -\epsilon_p \mathbf{R}_p^{-1} & 1 \end{bmatrix} \quad (\text{B.20})$$

and (B.18) is multiplied with this matrix \mathbf{F} . The result of the multiplication

$$\mathbf{R}_{p+1} [\mathbf{a}_{p+1}^1 \mathbf{b}_{p+1}^1] \mathbf{F} = [\mathbf{e}_{p+1}^1 \mathbf{r}_{p+1}^1] \mathbf{F}$$

is equal to (B.19), so

$$[\mathbf{a}_{p+1}^1 \mathbf{b}_{p+1}^1] \mathbf{F} = [\mathbf{a}_{p+1} \mathbf{b}_{p+1}]$$

gives the recursions

$$\begin{aligned} \mathbf{a}_{p+1} &= \mathbf{a}_{p+1}^1 - \epsilon_p \mathbf{R}_p^{-1} \mathbf{b}_{p+1}^1 \\ \mathbf{b}_{p+1} &= \mathbf{b}_{p+1}^1 - \epsilon_p^- \mathbf{E}_p^{-1} \mathbf{a}_{p+1}^1 \end{aligned} \quad (\text{B.21})$$

while

$$[\mathbf{e}_{p+1}^1 \mathbf{r}_{p+1}^1] \mathbf{F} = [\mathbf{e}_{p+1} \mathbf{r}_{p+1}]$$

gives the recursions

$$\begin{aligned} \mathbf{E}_{p+1} &= \mathbf{E}_p - \epsilon_p \mathbf{R}_p^{-1} \epsilon_p^- \\ \mathbf{R}_{p+1} &= \mathbf{R}_p - \epsilon_p^- \mathbf{E}_p^{-1} \epsilon_p. \end{aligned} \quad (\text{B.22})$$

By associating polynomials with vectors as $\mathbf{a}_p \leftrightarrow \sum_{i=0}^p a_{p,i} z^{-i} = A_p(z)$ and as $\mathbf{b}_p \leftrightarrow \sum_{i=0}^p b_{p,i} z^{-p+i} = B_p(z)$, then the $(p+1) \times 1$ column vectors \mathbf{a}_{p+1}^1 and \mathbf{b}_{p+1}^1 can be related to these polynomials as $\mathbf{a}_{p+1}^1 \leftrightarrow \sum_{i=0}^p a_{p,i} z^{-i} + 0 \times z^{-p-1} = A_p(z)$ and as $\mathbf{b}_{p+1}^1 \leftrightarrow 0 \times z^{-0} + \sum_{i=0}^p b_{p,i} z^{-p+i-1} = z^{-1} B_p(z)$, and (B.21) can be written in matrix form as

$$\begin{bmatrix} A_{p+1}(z) \\ B_{p+1}(z) \end{bmatrix} = \begin{bmatrix} 1 & -\epsilon_p \mathbf{R}_p^{-1} z^{-1} \\ -\epsilon_p^- \mathbf{E}_p^{-1} & z^{-1} \end{bmatrix} \begin{bmatrix} A_p(z) \\ B_p(z) \end{bmatrix} \quad (\text{B.23})$$

If we now assume the matrix \mathbf{R}_p to be symmetric, that is $R(-i) = R(i)$ for $i = 1, 2, \dots, p$, then we find from (B.14) that $\sum_{j=0}^p a_{p,j} R(j-i) = \sum_{j=0}^p a_{p,j} R(i-j) = 0$ for $i = 1, 2, \dots, p$. From (B.15) we get the equations $\sum_{j=0}^p b_{p,j} R(i-j) = 0$ and comparing this with the previous results we conclude that

$$a_{p,j} = b_{p,j} \text{ for } j = 1, 2, \dots, p.$$

This results in the next equalities

$$\begin{aligned} \epsilon_p &= \epsilon_p^-, \\ \mathbf{E}_p &= \mathbf{R}_p, \\ \mathbf{b}_p &= \mathbf{a}_p^r. \end{aligned} \quad (\text{B.24})$$

The last equation gives as the associated polynomial

$$B_p(z) = \sum_{i=0}^p a_{p,i} z^{-p+i} = z^{-p} A_p(z^{-1}) = A_p^r(z)$$

and the recursion (B.23) changes into (2.22) if the parameter γ_{p+1} , as defined in (B.11), is used. Also the relations (B.8) and (B.12) are valid in the symmetric situation.

NOTE :

In the non-symmetric Toeplitz situation it is important to notice how several parameters are defined. Here we show the influence of the choice of the matrix \mathbf{R}_p and the interpretation of the expression $\mathbb{E}[y_{n-i} y_{n-j}]$ on the form of the Yule-Walker equation (YWE). To be exact it is repeated that the vectors \mathbf{a}_p and \mathbf{e}_p are column vectors.

i. For the matrix \mathbf{R}_p the elements $R_{i,j} = R(j-i)$ are used for $0 \leq i, j \leq p$.

- If $\mathbb{E}[y_{n-i} y_{n-j}] = R(j-i)$ then the YWE becomes $\mathbf{a}_p^T \mathbf{R}_p = \mathbf{e}_p^T$,

- if $\mathbb{E}[y_{n-i}y_{n-j}] = R(i-j)$ then the YWE becomes $\mathbf{R}_p \mathbf{a}_p = \mathbf{e}_p$.
- ii. For the matrix \mathbf{R}'_p the elements $R'_{i,j} = R(i-j)$ are used for $0 \leq i, j \leq p$.
- If $\mathbb{E}[y_{n-i}y_{n-j}] = R(j-i)$ then the YWE becomes $\mathbf{a}_p^T \mathbf{R}'_p = \mathbf{e}_p^T$ [12] ,
 - if $\mathbb{E}[y_{n-i}y_{n-j}] = R(i-j)$ then the YWE becomes $\mathbf{R}'_p \mathbf{a}_p^T = \mathbf{e}_p^T$.

Similar expressions for the backward YWE are valid if \mathbf{a}_p and \mathbf{e}_p are replaced by \mathbf{b}_p and \mathbf{r}_p respectively.

For notation reasons, the method of the second line of i. is used in this appendix. When the matrix is symmetric all the expressions of the YWE become the same, because $\mathbf{R}_p = \mathbf{R}'_p = \mathbf{R}_p^T$ and because of (B.3) and (B.4).

B.3 The physical meaning of several quantities

As in section 2.2 we make a forward prediction for y_n from the p sample values in the past and the prediction error becomes $e_p(n) = \sum_{i=0}^p a_{p,i} y_{n-i}$ with $a_{p0} = 1$. Because the error is orthogonal to y_{n-i} , so $\mathbb{E}[e_p(n)y_{n-i}] = 0$ for $1 \leq i \leq p$, we have

$$\sum_{j=0}^p a_{p,j} \mathbb{E}[y_{n-j}y_{n-i}] = \sum_{j=0}^p a_{p,j} R(j-i) = 0, \quad 1 \leq i \leq p. \quad (\text{B.25})$$

For the mean-square value of the prediction error we found

$$E_p = \mathbb{E}[e_p^2(n)] = \sum_{i=0}^p a_{p,i} \mathbb{E}[y_{n-i}y_n] = \sum_{i=0}^p a_{p,i} R(i). \quad (\text{B.26})$$

If a backward prediction is made for y_{n-p} from the p values from the future, the prediction error in this case is $r_p(n) = \sum_{i=0}^p b_{p,i} y_{n-p+i}$, which is orthogonal with y_{n-p+i} , so $\mathbb{E}[r_p(n)y_{n-p+i}] = 0$ for $1 \leq i \leq p$ and with $b_{p0} = 1$. Or

$$\sum_{j=0}^p b_{p,j} \mathbb{E}[y_{n-p+j}y_{n-p+i}] = \sum_{j=0}^p b_{p,j} R(i-j) = 0, \quad 1 \leq i \leq p. \quad (\text{B.27})$$

The mean-square error is now

$$R_p = \mathbb{E}[r_p^2(n)] = \sum_{j=0}^p b_{p,j} \mathbb{E}[y_{n-p+j}y_{n-p}] = \sum_{j=0}^p b_{p,j} R(-j). \quad (\text{B.28})$$

The formulas (B.25) to (B.28) can be redefined in the double Yule-Walker equation

$$\mathbf{R}_p[\mathbf{a}_p \mathbf{b}_p] = [\mathbf{e}_p \mathbf{r}_p].$$

The forward prediction is made from the sequence $\{y_{n-1}, \dots, y_{n-p}\}$, while the backward prediction is made from the sequence $\{y_n, \dots, y_{n-p+1}\}$. We now make a backward prediction

with the same sequence as we used for the forward prediction. These prediction for y_{n-1-p} becomes $b_{p1}y_{n-p} + b_{p2}y_{n-p+1} + \dots + b_{pp}y_{n-1}$ and the prediction error is

$$r_p(n-1) = \sum_{i=0}^p b_{p,i} y_{n-1-p+i}.$$

If the correlation between this backward prediction error and the forward prediction error is investigated, the following relations are found

$$\begin{aligned} \mathbb{E}[r_p(n-1)e_p(n)] &= \sum_{i=0}^p b_{p,i} \sum_{j=0}^p a_{p,j} R(-j-i+p+1) \\ &= \sum_{j=0}^p a_{p,j} R(-j+p+1) + \sum_{i=1}^p b'_{p,i} \sum_{j=0}^p a_{p,j} R(i-j) \\ \mathbb{E}[e_p(n)r_p(n-1)] &= \sum_{j=0}^p b_{p,j} R(j-p-1) + \sum_{i=1}^p a'_p i j \sum_{j=0}^p b_{p,j} R(j-i). \end{aligned}$$

For the symmetric situation these become

$$\mathbb{E}[r_p(n-1)e_p(n)] = \epsilon_p + \sum_{i=1}^p b'_{p,i} \sum_{j=0}^p a_{p,j} R(j-i) = \epsilon_p$$

and

$$\mathbb{E}[e_p(n)r_p(n-1)] = \epsilon_p^- + \sum_{i=1}^p a'_{p,i} \sum_{j=0}^p b_{p,j} R(i-j) = \epsilon_p,$$

where (B.17), (B.24), (B.25) and (B.27) are used. By using (B.11), we see that the PARCOR coefficient is the partial correlation between the forward and backward prediction error, or

$$\gamma_{p+1} = \frac{\epsilon_p}{\bar{E}_p} = \frac{\mathbb{E}[r_p(n-1)e_p(n)]}{\mathbb{E}[e_p^2(n)]} = \frac{\mathbb{E}[r_p(n-1)e_p(n)]}{\mathbb{E}[r_p^2(n-1)]}.$$

Appendix C

The Cholesky decomposition

The covariance method requires the solution of a set of simultaneous linear equations, which may be generally expressed as a matrix equation

$$\mathbf{Ax} = \mathbf{b},$$

where \mathbf{A} is some arbitrary $n \times n$ square matrix, \mathbf{b} is some arbitrary $n \times 1$ column vector and \mathbf{x} is an $n \times 1$ column vector with unknown components whose solution is to be found. The Gaussian elimination process may be used using three steps

1. The matrix \mathbf{A} is factored into a product of an upper triangular matrix \mathbf{U} and a lower triangular matrix \mathbf{L} (with 1's along the diagonal)

$$\mathbf{A} = \mathbf{LU}.$$

2. The first back substitution finds the triangular matrix solution of

$$\mathbf{Ly} = \mathbf{b}.$$

3. The second back substitution by the triangular matrix solution for the \mathbf{x} vector

$$\mathbf{Ux} = \mathbf{y}.$$

If the matrix \mathbf{A} is square and symmetric the triangular factorization takes on the special form

$$\mathbf{A} = \mathbf{LL}^T.$$

The upper triangular matrix is the transpose of the lower triangular matrix, so one matrix has to be determined. This decomposition is called the Cholesky decomposition. For the normal definition of transpose, the elements $l_{j,i}$ for $1 \leq j \leq n$ and $j \leq i \leq n$ of \mathbf{L}^T are equal to the terms $l_{i,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq i$ of the matrix \mathbf{L} .

Now we will give more details about the three steps of the Cholesky decomposition. In the **first** step the lower triangular matrix has to be determined from the following matrix equation:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} = \begin{bmatrix} l_{1,1} & & & \\ l_{2,1} & l_{2,2} & & \\ \vdots & \vdots & \ddots & \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix} \begin{bmatrix} l_{1,1} & l_{2,1} & \cdots & l_{n,1} \\ & l_{2,2} & \cdots & l_{n,2} \\ & & \ddots & \\ & & & l_{n,n} \end{bmatrix}.$$

From this equation it is easy to see that the next equations are valid

$$\begin{aligned}
a_{1,1} &= l_{1,1}^2; \quad j = 1, i = 1 \\
a_{i,1} &= l_{i,1}l_{1,1}; \quad j = 1, 2 \leq i \leq n \\
a_{i,j} &= \sum_{k=1}^j l_{i,k}l_{j,k} = \sum_{k=1}^{j-1} l_{i,k}l_{j,k} + l_{i,j}l_{j,j}; \quad 2 \leq j \leq i-1, 2 \leq i \leq n \\
a_{i,i} &= \sum_{k=1}^{i-1} l_{i,k}^2 + l_{i,i}^2; \quad 2 \leq i \leq n.
\end{aligned}$$

Now the components $l_{i,j}$ of the lower triangular matrix can be found as

$$\begin{aligned}
l_{1,1} &= \sqrt{a_{1,1}}; \quad j = 1, i = 1 \\
l_{i,1} &= \frac{a_{i,1}}{l_{1,1}}; \quad j = 1, 2 \leq i \leq n \\
l_{i,j} &= \frac{1}{l_{j,j}}[a_{i,j} - \sum_{k=1}^{j-1} l_{i,k}l_{j,k}]; \quad 2 \leq j \leq i-1, 2 \leq i \leq n \\
l_{i,i} &= \sqrt{a_{i,i} - \sum_{k=1}^{i-1} l_{i,k}^2}; \quad 2 \leq i \leq n.
\end{aligned}$$

In the **second** step the components y_i has to be found from the next matrix equation

$$\begin{bmatrix} l_{1,1} & & & \\ l_{2,1} & l_{2,2} & & \\ \vdots & \vdots & & \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

and are as follows

$$\begin{aligned}
y_1 &= \frac{b_1}{l_{1,1}} \\
y_i &= \frac{1}{l_{i,i}}[b_i - \sum_{k=1}^{i-1} l_{i,k}y_k]; \quad 2 \leq i \leq n.
\end{aligned}$$

In the **third** step the elements x_i of the vector x are determined from the next matrix equation

$$\begin{bmatrix} l_{1,1} & l_{2,1} & \cdots & l_{n,1} \\ & l_{2,2} & \cdots & l_{n,2} \\ & & & \vdots \\ & & & l_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

The solution is

$$\begin{aligned}
x_n &= \frac{y_n}{l_{n,n}} \\
x_i &= \frac{1}{l_{i,i}}[y_i - \sum_{k=i+1}^n l_{k,i}x_k]; \quad 1 \leq i \leq n-1.
\end{aligned}$$

It seems usefull to introduce a modified definition for the transpose of the lower matrix in such a manner that the elements on the diagonal of the upper triangular matrix are one. So the elements of the upper triangular matrix become $u_{j,j} = 1$ for $1 \leq j \leq n$ and $u_{j,i} = \frac{l_{i,j}}{l_{j,j}}$ for $1 \leq j \leq n$ and $j < i \leq n$, where $l_{i,j}$ are the elements of the lower triangular matrix. In this case the **first** step implies the solution of the elements $l_{i,j}$ from the next matrix equation:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} = \begin{bmatrix} l_{1,1} & & & \\ l_{2,1} & l_{2,2} & & \\ \vdots & \vdots & & \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix} \begin{bmatrix} 1 & \frac{l_{2,1}}{l_{1,1}} & \cdots & \frac{l_{n,1}}{l_{1,1}} \\ & 1 & \cdots & \frac{l_{n,2}}{l_{2,2}} \\ & & \ddots & \\ & & & 1 \end{bmatrix}.$$

The elements are now

$$\begin{aligned} l_{i,1} &= a_{i,1}; \quad j = 1, \quad 1 \leq i \leq n \\ l_{i,j} &= a_{i,j} - \sum_{k=1}^{j-1} \frac{l_{i,k} l_{j,k}}{l_{k,k}}; \quad 2 \leq j \leq i, \quad 1 \leq i \leq n \end{aligned}$$

The values for y_i in the **second** step are the same as in the previous situation but with the values of $l_{i,j}$ as given above, while the x_i of the **third** step are

$$\begin{aligned} x_n &= y_n \\ x_i &= y_i - \frac{1}{l_{i,i}} \sum_{k=i+1}^n l_{k,i} x_k; \quad 1 \leq i \leq n-1. \end{aligned}$$

Appendix D

Procedures in TURBO PASCAL

In this appendix we describe the TURBO-PASCAL procedures belonging to the algorithms given in this report. It is assumed that a mathematical coprocessor is available, because the type single is used. If the coprocessor is not available, the statement

```
type single = real ;
```

is sufficient to change the types. First some constants and types are introduced. The constant Mmax is the maximum order of the predictor, N is the amount of (speech) data available. The type data is given to arrays containing (speech) data as floating point values, while the type autocor is given to arrays containing several kinds of results, such as autocorrelation functions ($R(0), R(1), \dots, R(M)$), the prediction or reflection coefficients or the coefficients of the $F(z)$ or $Q(z)$ polynomials.

```
const N = 200 ;           {# of samples in speech frame}
      Nm1 = N-1 ;
      Mmax = 20 ;         {maximum order of prediction}
      MMmax = Mmax+1 ;
type data = array[0..Nm1] of single ;
      autocor = array[0..MMmax] of single ; {M+1 for F & Q polynomial}
```

The two following procedures, **auto1** and **auto2**, determine the sample autocorrelation function. Both procedures have as input an array, y, containing the (speech) data and two local variables, N1 and M1, as information about the amount of (speech) data and the model order. The output is in both cases an array, R, containing the autocorrelation function. The procedure auto1 calculates the autocorrelation function straight forward according to (5.1). The procedure auto2 is more efficient for those computers which calculate a summation more faster than a multiplication because nearly half the number of multiplications is replaced by additions. This is obtained by the factorization of (5.1) shown in the next example for $p = 3$

$$\begin{aligned} R(3) &= y_0 y_3 + y_1 y_4 + y_2 y_5 + y_3 y_6 + y_4 y_7 + y_5 y_8 + \dots \\ &= (y_0 + y_6) y_3 + (y_1 + y_7) y_4 + (y_2 + y_8) y_5 + \dots \end{aligned}$$

For a *good* working procedure auto2 the order of the model, M1, must be smaller or equal than the amount of data, N1, divided by three, or

$$M1 \leq \frac{N1}{3}.$$

```

PROCEDURE auto1 ( var y:data; var R:autocor; N1, M1 : integer ) ;
var k, i : integer ;
    rr : double ;
begin
    for k := 0 to M1 do
        begin
            rr := 0 ;
            for i := 0 to N1-1-k do rr := rr + y[i] * y[i+k] ;
            R[k] := rr ;
        end ;
    end ; { end of auto1 }

```

```

PROCEDURE auto2 ( var y:data; var R:autocor; N1, M1 : integer ) ;
(* WARNING ----- ONLY FOR N1 >= 3*M1 !!!! ----- *)
var i, j, imod, nterm,
    p, pl, pr, pstrt, pstop : integer ;
    rr : double ;
begin
    rr := 0 ;
    for i := 0 to N1 - 1 do rr := rr + y[i] * y[i] ;
    R[0] := rr ;
    for i := 1 to M1 do
        begin
            imod := 2 * i ;
            nterm := N1-1-i ;
            rr := 0 ;
            for j := i to imod-1 do
                begin
                    pr := j - i ;
                    p := j ;
                    repeat
                        pl := pr ;
                        pr := p + i ;
                        rr := rr + y[p] * ( y[pl] + y[pr] ) ;
                        p := p + imod ;
                    until p > nterm ;
                end ;
            if pr <> N1 - 1 then
                begin
                    if (N1-1-pr) < i then
                        begin
                            pstrt := pl + i + 1 ;
                            pstop := nterm ;
                        end
                    else
                        begin
                            pstrt := N1 - imod ;

```

```

        pstop := pr ;
    end ;
    for p := pstrt to pstop do rr := rr + y[p] * y[p+i] ;
    end ;
    R[i] := rr ;
end ;
end ; { end of auto2 }

```

The procedure **Levinson** has as input the array with the autocorrelation function, **R**, and the model order, **M1**. The outputs are the array **rc**, filled with the reflection coefficients, and the array **a**, containing the prediction coefficients. The minimal value of the prediction error energy is given by **rc[0]**.

```

PROCEDURE Levinson ( var R, rc, a : autocor; M1 : integer ) ;
var p, ip, iph, mh : integer ;
    delta, at : single ;
begin
    rc[1] := R[1]/R[0] ;      a[1] := - rc[1] ;
    rc[0] := R[0] - R[1]*rc[1] ; a[0] := 1 ;
    for p := 2 to M1 do
    begin
        delta := 0 ;
        for ip := 0 to p-1 do delta := delta + R[p-ip]*a[ip] ;
        rc[p] := delta/rc[0] ;
        mh := trunc(p/2) ;
        for ip := 1 to mh do
        begin
            iph := p - ip ;
            at := a[ip] - rc[p]*a[iph] ;
            a[iph] := a[iph] - rc[p]*a[ip] ;
            a[ip] := at ;
        end ;
        a[p] := - rc[p] ;
        rc[0] := rc[0] -rc[p]*delta ;
    end ;
end ; { end of Levinson }

```

The procedure **Split_Levinson** has the same inputs as the procedure **Levinson**, but the output is here the reflection coefficients array only. If the **a**-parameters are also wanted an array, **a**, can be added to the parameter list. For further instructions see the directions at the bottom of the procedure.

```

PROCEDURE Split_Levinson ( var R, rc : autocor; M1 : integer ) ;
var p, ip, nterm1, nterm2 : integer ;
    tau, tau_prev, alpha, fh : single ;
    sum1, sum2, lambda : single ;
    f, fa, a : array[0..Mmax] of single ;
begin

```

```

f[0] := 1 ; fa[0] := 1 ;
rc[1] := R[1]/R[0] ;
f[1] := -2*rc[1] ;
tau := R[0] + R[1] ;
for p := 2 to M1 do
begin
    tau_prev := tau ;
    tau := 0 ;
    if odd(p) then nterm1 := trunc((p+1)/2)
        else nterm1 := trunc(p/2) ;
    for ip := 0 to nterm1-1 do tau := tau + ( R[ip]+R[p-ip])*f[ip] ;
    if not odd(p) then tau := tau + R[trunc(p/2)]*f[trunc(p/2)] ;
    alpha := tau/tau_prev ;
    rc[p] := -1 + alpha/(1-rc[p-1]) ;    {----- LINE 1 -----}
{----- BLOCK 1 -begin-----}
    if p <> M1 then
    begin
        for ip := nterm1 downto 1 do
        begin
            fh := f[ip] ;
            f[ip] := f[ip] + f[ip-1] - alpha*fa[ip-1] ;
            fa[ip] := fh ;
            if (not odd(p)) and (ip = nterm1) then f[nterm1+1] := f[ip] ;
        end ;
    end ;    {----- BLOCK 2 -end-----}
{-----}
    end ;
    rc[0] := ( 1- rc[M1] ) * tau ;    {----- LINE 2 -----}
{-----}
(* if odd(M1) then nterm2 := trunc((M1-1)/2) {----- BLOCK 2 -begin-----}
    else nterm2 := trunc(M1/2) ;
sum1 := 2 ; sum2 := 2 ;
for ip := 1 to nterm2 do
begin
    sum1 := sum1 + 2*f[ip] ;
    sum2 := sum2 + 2*fa[ip] ;
end ;
if odd(M1) then sum1 := sum1 + f[nterm2+1]
    else sum2 := sum2 - fa[nterm2] ;
lambda := sum1/sum2 ;
a[0] := 1 ;
for ip := 1 to M1 do
begin
    if ip <= nterm1
        then a[ip] := a[ip-1] + f[ip] - lambda * fa[ip-1]
        else a[ip] := a[ip-1] + f[M1-ip+1] - lambda * fa[M1-ip+1] ;
end ; *)    {----- BLOCK 2 -end-----}

```

```

{-----}
(*-----
For Reflection Coefficients :
    1) line 1 on
    2) block 1 on for  $2 < p < M1$ 
    3) line 2 on
    4) block 2 off
For Predictor Coefficients :
    1) line 1 off
    2) block 1 on for  $2 < p \leq M1$ 
    3) line 2 off
    4) block 2 on
-----*)
end ; { end of Split_Levinson }

```

If a procedure give as output the reflection coefficients only and the prediction coefficients are needed, the procedure **StepUp** can be used. This procedure fills an array, a, with prediction coefficient from an array, rc, containing the reflection coefficient with (2.19). The procedure **StepDown** does the reverse, it calculates the reflection coefficients from the a-parameters.

```

PROCEDURE StepUp ( var rc, a : autocor; M1 : integer ) ;
var p, ip : integer ;
    b : array[1..Mmax] of single ;
begin
    a[0] := 1 ; a[1] := - rc[1] ;
    for p := 2 to M1 do
        begin
            for ip := 1 to p-1 do b[ip] := a[p-ip] ;
            for ip := 1 to p-1 do a[ip] := a[ip] - rc[p]*b[ip] ;
            a[p] := - rc[p] ;
        end ;
    end ; { end of StepUp }

PROCEDURE StepDown ( var a, rc : autocor; M1 : integer ) ;
{pre : array a contains the coefficients of the predictor A(z)
 post: array rc contains the reflection-coefficients }
var
    p, ip : integer;
    b : array[1..Mmax] of single;
    den : single; { help variable for storing the denominator }

begin
    for p := M1 downto 2 do
        begin
            rc[p] := - a[p]; den := 1 - rc[p] * rc[p];
            for ip := 1 TO p-1 do b[ip] := a[p-ip];
            for ip := 1 TO p-1 do a[ip] := ( a[ip] + rc[p] * b[ip] ) / den;
        end;
    end;

```



```

    rc[1] := - a[1];
end; { end of StepDown }

```

The procedures, **Analysis** and **Synthesis**, simulate the analysis and the synthesis lattice filters respectively.

```

PROCEDURE Analysis ( input : single; var rc : autocor; var output : single;
                    M1 : integer ) ;

```

```

var i : integer ;
    ep, emh, emhh : single ;
    em : array[1..Mmax] of single ; { must be initialized !!!! }
begin
    ep := input ; emhh := input ;
    for i := 1 to M1 do
    begin
        emh := em[i] - rc[i] * ep ;
        ep := ep - rc[i] * em[i] ;
        em[i] := emhh ;
        emhh := emh ;
    end ;
    output := ep ;
end ; { end of Analysis }

```

```

PROCEDURE Synthesis ( input : single; var rc : autocor; var output : single;
                    M1 : integer ) ;

```

```

var i : integer ;
    ep : single ;
    em : array[1..Mmax] of single ; { Initialize !!! }
begin
    ep := input + rc[M1] * em[M1] ;
    for i := M1-1 downto 1 do
    begin
        ep := ep + rc[i] * em[i] ;
        em[i+1] := em[i] - rc[i] * ep ;
    end ;
    em[1] := ep ;
    output := ep ;
end ; { end of Synthesis }

```

The procedures, **Schur** and **Split_Schur**, determine the reflection coefficients in the array, rc, from the autocorrelation function in the array R.

```

PROCEDURE Schur ( var R, rc : autocor; M1 : integer ) ;

```

```

var k, p : integer ;
    gamma, temp : single ;
    g, gr : array[0..Mmax] of single ;
begin
    for k := 0 to M1 do

```

```

begin
  g[k] := R[k] ;
  gr[k] := r[k] ;
end ;
for p := 0 to M1-1 do
begin
  gamma := g[p+1] / gr[p] ;
  for k := M1 downto p+1 do
  begin
    temp := g[k] ;
    g[k] := temp - gamma * gr[k-1] ;
    gr[k] := gr[k-1] - gamma * temp ;
  end ;
  rc[p+1] := gamma ;
end ;
rc[0] := gr[M1] ;
end ; { end of Schur }

PROCEDURE Split_Schur ( var R, rc : autocor; M1 : integer ) ;
var l : array[0..Mmax] of array[0..Mmax] of single ;
    k, p : integer ;
    gamma, alpha : single ;
begin
  l[0,0] := R[0] ;      { l[0,0] = tau0 }
  for k := 1 to M1 do
  begin
    l[0,k] := 2 * R[k] ;
    l[1,k] := R[k] + R[k-1] ;
  end ;
  gamma := 0 ;
  for p := 1 to M1-1 do
  begin
    alpha := l[p,p]/l[p-1,p-1] ;
    gamma := -1 + alpha/(1-gamma) ;
    rc[p] := gamma ;
    for k := p+1 to M1 do
      l[p+1,k] := l[p,k] + l[p,k-1] - alpha*l[p-1,k-1] ;
    end ;
    alpha := l[M1,M1]/l[M1-1,M1-1] ;
    rc[M1] := -1 + alpha/(1-gamma) ;
    rc[0] := l[M1,M1]*(1-rc[M1]) ;
  end ; { end of Split_Schur }

```

The procedure **make_fq** determine the coefficients of the polynomials $F_{M+1}(z)$ and $Q_{M+1}(z)$ from the prediction coefficients with the relations (4.2). The input of the procedure is the array, **a**, containing the prediction coefficients and the variable **M1**. The outputs are two arrays, **f_pol** and **q_pol**, containing the coefficients of the two polynomials. The two arrays

are arranged in the following way

$$\begin{aligned} f_pol[i] &= f_{M+1,i} \\ q_pol[i] &= q_{M+1,i} \text{ for } i = 0, 1, \dots, M+1. \end{aligned}$$

```

PROCEDURE Make_fq ( var a, f_pol, q_pol : autocor; M1 : integer) ;
var i, mp1, nterm : integer ;
begin
  mp1 := M1+1 ;
  nterm := trunc(mp1/2) ;
  f_pol[0] := 1 ; f_pol[mp1] := 1 ;
  q_pol[0] := 1 ; q_pol[mp1] := -1 ;
  for i := 1 to nterm-1 do
    begin
      f_pol[i] := a[i] + a[mp1-i] ;
      f_pol[mp1-i] := f_pol[i] ;
      q_pol[i] := a[i] - a[mp1-i] ;
      q_pol[mp1-i] := -q_pol[i] ;
    end ;
  if odd(mp1) then
    begin
      f_pol[nterm] := a[nterm] + a[mp1-nterm] ;
      f_pol[mp1-nterm] := f_pol[nterm] ;
      q_pol[nterm] := a[nterm] - a[mp1-nterm] ;
      q_pol[mp1-nterm] := -q_pol[nterm] ;
    end
  else
    begin
      f_pol[nterm] := a[nterm] + a[mp1-nterm] ;
      q_pol[nterm] := 0 ;
    end ;
  end ; { end of Make_fq }

```

The next two procedures use procedures from [21], so a new constant and three new types are introduced. These new constant and types are needed for the routines zroots and hqr.

```

const TMMmax = 2*MMmax ;
type glnp      = array[1..MMmax] of single ;
  glnpnp      = array[1..MMmax,1..MMmax] of single ;
  glcarray    = array[1..TMMmax] of single ;

```

The procedures **Roots_zr** and **Roots_com** determine the zero's or roots of a polynomial of order M1. It is assumed that the polynomial is from the type with negative exponents in z , thus

$$Pol(z) = \sum_{i=0}^{M1} x_pol[i] z^{-i}.$$

The input of the procedures is an array, x_pol , containing the (real) coefficients of the polynomial. The outputs are two array, $real_root$ and $imag_root$, containing the real and imaginary part of the roots.

```

PROCEDURE Roots_zr ( var x_pol : autocor; var real_root, imag_root : glnp;
                    M1 : integer ) ;
var
  i, n : integer;
  a, y : glcarray;
begin
  for i := 0 to M1 do
    begin
      a[2*i+1] := x_pol[M1-i] ; { reverse the coefficients, if polynomial ..}
      a[2*i+2] := 0 ;           { .. is NOT symmetric }
    end ;
    zroots( a, M1, y, true) ;
    for n := 1 to M1 do
      begin
        real_root[n] := y[2*n-1] ;
        imag_root[n] := y[2*n] ;
      end ;
    end ; { end of Roots_zr }

```

```

PROCEDURE Roots_com ( var x_pol : autocor ; var real_root, imag_root : glnp;
                     M1 : integer ) ;
var i, j : integer ;
    compan : glnpnp ;
begin
  for i := 1 to M1 do for j := 1 to order do compan[i,j] := 0 ;
  for i := 1 to M1 do compan[i,i] := -x_pol[i] ; { reverse the coefficients}
  for i := 2 to M1 do compan[i,i-1] := 1 ;      { companion matrix ready }
  hqr ( compan, M1, real_root, imag_root ) ;    { Hessenberg matrix }
end ; { end of Roots_com }

```

The procedure **Roots_cheb** determines the roots of $F_{M+1}(z)$ and of $Q_{M+1}(z)$ by searching for zero's of the functions $F'(x)$ and $Q'(x)$ over the interval $[-1,1]$ of x . The output array, **Roots**, contains with increasing address the values of x , for which the functions $F'(x)$ and $Q'(x)$ alternately have zero's. At address 1 is the highest (real) value of x from $F'(x)$, at address 2 the highest (real) value from $Q'(x)$ and so on.

```

PROCEDURE Roots_cheb( var F_pol, Q_pol : autocor; var Roots : glnp;
                     M1 : integer; var Numfound : integer ) ;
CONST
  Delta = 0.02; { step size for the search over the interval [-1,1] }
  NumBis = 16;  { number of bisections for the determination of the .. }
                  { ..positions of the zero's }
VAR
  M1,M2      : integer; { orders F'(x) resp. Q'(x) }
  cf, cq     : autocor; { coefficients c_i of F'(x) and c'_i of Q'(x) }

FUNCTION Sign_F ( x : real ): boolean;
{ calculates the sign of chebyshev-polynomial F'(x) at point x }

```

```

{ returns (  $F'(x) > 0$  ) }
VAR
  b          : autocor; { coefficients  $b_i$  }
  i          : integer; { counter }
BEGIN
  { initialize }
  b[M1+1] := 0.0; b[M1+2] := 0.0;
  i := M1; { start at the highest power of  $F'(x)$  }
  WHILE (i >= 0) DO BEGIN { determination of  $b_0$  and  $b_2$  }
    b[i] := cf[i] - b[i+2] + 2 * x * b[i+1];
    i := i - 1;
  END;
  Sign_F := ((b[0] - b[2] + cf[0]) > 0);
END; { Sign_F }

FUNCTION Sign_Q ( x : real ): boolean;
{ calculates the sign of chebyshev-polynomial  $Q'(x)$  at point x }
{ returns (  $Q'(x) > 0$  ) }
VAR
  b          : autocor; { coefficients  $b'_i$  }
  i          : integer; { counter }
BEGIN
  { initialize }
  b[M2+1] := 0.0; b[M2+2] := 0.0;
  i := M2; { start at the highest power of  $Q'(x)$  }
  WHILE (i >= 0) DO BEGIN { determination of  $b_0$  and  $b_2$  }
    b[i] := cq[i] - b[i+2] + 2 * x * b[i+1];
    i := i - 1;
  END;
  Sign_Q := ((b[0] - b[2] + cq[0]) > 0);
END; { Sign_Q }

VAR
  fun          : Boolean; { which function is on turn: TRUE=F / FALSE=Q }
  lastsign     : Boolean;
  i            : integer; { counter }
  f,q          : autocor; { intermediate results (coeff. of  $F(z)$  and  $Q(z)$ ) }
  x, xmid, lastx : single;
  rootnum      : integer; { number of roots found }

BEGIN
  { Initialize }
  M1 := M1 DIV 2 + M1 MOD 2; { order  $F'(x)$  }
  M2 := M1 DIV 2; { order  $Q'(x)$  }
  rootnum := 0; { no root found at this moment }

  { calculate  $c_i$  and  $c'_i$  (cf and cq) via  $f_i$  and  $q_i$  }

```

```

f[0] := 1.0;
q[0] := 1.0;
IF ( M1 MOD 2 = 0 ) THEN BEGIN
  FOR i := 1 to M1 do f[i] := F_pol[i] - f[i-1];
  FOR i := 1 to M2 do q[i] := Q_pol[i] + q[i-1];
END
ELSE BEGIN
  FOR i := 1 to M1 do f[i] := F_pol[i];
  q[1] := Q_pol[1];
  FOR i := 2 to M2 do q[i] := Q_pol[i] + q[i-2];
END;
cf[0] := f[M1];
for i := 1 to M1-1 do cf[i] := f[M1-i] + f[M1-i];
cf[M1] := 2.0; { follows direct from f[0] }
cq[0] := q[M2];
for i := 1 to M2-1 do cq[i] := q[M2-i] + q[M2-i];
cq[M2] := 2.0; { follows direct from q[0] }

{ Initialize the search for zero's }
x := 1.0; { startpoint: x = 1 }
fun := true; { Start with F'(x), because first zero in F'(x) }
lastsign := Sign_F(x); lastx := 1.0; { calculate sign F'(x) in startpoint }
x := x - Delta;
{ search for zero's }
WHILE (lastx > -1.0) AND (rootnum < M1) DO BEGIN
{ just in interval [-1,1] are zero's ( maximal M ) }
  IF fun THEN BEGIN { search in F'(x) }
    IF Sign_F(x) <> lastsign THEN BEGIN
      { Interval found that contains zero }
      FOR i := 1 to NumBis DO BEGIN { Bisection }
        xmid := (x + lastx)/2; { middle of section }
        IF Sign_F(xmid) = lastsign THEN lastx := xmid ELSE x := xmid;
      END; { Bisection }
      x := (x + lastx)/2; { middle of last section }
      rootnum := rootnum + 1;
      Roots[rootnum] := x;
      fun := not fun; { next zero in Q'(x) }
      lastsign := Sign_Q(x); {calculate sign of Q'(x) with zero of..}
    END; { Bisection F' in interval }          {...F'(x) as startvalue}
  END { search in F'(x) }
  ELSE BEGIN { search in Q'(x) }
    IF Sign_Q(x) <> lastsign THEN BEGIN
      { Interval found that contains zero }
      FOR i := 1 to NumBis DO BEGIN { Bisection }
        xmid := (x + lastx)/2; { middle of section }
        IF Sign_Q(xmid) = lastsign THEN lastx := xmid ELSE x := xmid;
      END; { Bisection }
    END;
  END;
END;

```

```

    x := (x + lastx)/2; { middle of last section }
    rootnum := rootnum + 1;
    Roots[rootnum] := x;
    fun := not fun; { next zero in F'(x) }
    lastsign := Sign_F(x); {calculate sign of F'(x) with zero of..}
    END; { Bisection Q' in interval }          {...Q'(x) as startvalue}
END; { search in Q'(x) }
lastx := x; x := x - Delta; { shift an interval }
END;
Numfound := rootnum; { number of zero's found for output }
END;{ end of Roots_cheb }

```

The procedure **Make_A** reconstructs the a-parameters, in the array, a, from the zero's of the polynomials $F'(x)$ and $Q'(x)$, obtained from the previous procedure (**Roots_cheb**). The content of the array, **Roots**, must be in the same order as described for the previous procedure.

```

PROCEDURE Make_A ( var Roots : glnp; var a : autocor; M1 : integer ) ;
VAR
  i, K : integer; { counters }
  f,q,g : autocor; { polynomial coeff. }
  M1, M2 : integer; { Order of F'(x) resp. Q'(x) }

BEGIN
  M1 := M1 DIV 2 + M1 MOD 2;
  M2 := M1 DIV 2;
  { determine coeff. f_i of F(z) }
  f[0] := 1.0; g[0] := 0.0;
  f[1] := -2 * Roots[1];
  IF (M1 > 1) THEN
    FOR K := 1 TO M1-1 DO BEGIN
      FOR i := 0 TO K DO g[i+1] := f[i];
      f[K+1] := - 2 * Roots[2*K+1] * g[K+1] + 2 * g[K];
      FOR i := 1 TO K DO
        f[K+1-i] := g[K+2-i] - 2 * Roots[2*K+1] * g[K+1-i] + g[K-i];
      END;
    END;
  FOR i := 0 TO M1-1 DO f[2*M1-i] := f[i];
  { determine coeff. q_i of Q(z) }
  q[0] := 1.0; g[0] := 0.0;
  IF (M2 > 0) THEN q[1] := -2 * Roots[2];
  IF (M2 > 1) THEN
    FOR K := 1 TO M2-1 DO BEGIN
      FOR i := 0 TO K DO g[i+1] := q[i];
      q[K+1] := - 2 * Roots[2*(K+1)] * g[K+1] + 2 * g[K];
      FOR i := 1 TO K DO
        q[K+1-i] := g[K+2-i] - 2 * Roots[2*(K+1)] * g[K+1-i] + g[K-i];
      END;
    END;
  IF (M2 > 0) THEN FOR i := 0 TO M2-1 DO q[2*M2-i] := q[i];

```

```

{ determine coeff. a_i of A(z) from F(z) and Q(z) }
a[0] := 1.0;
IF (M1 MOD 2 = 0) THEN { M1 even }
  FOR i := 1 TO M1 DO a[i] := (f[i-1] + f[i] - q[i-1] + q[i]) / 2
ELSE BEGIN { M1 odd }
  a[1] := (f[1] + q[1]) / 2;
  IF (M1 > 1) THEN FOR i := 2 TO M1 DO a[i] := (f[i] - q[i-2] + q[i]) / 2;
END;
END; { end of Make_A }

```

For the next procedures we need another type for the covariance matrix. So we introduce

```
type covar = array[0..Mmax,0..Mmax] of single ;
```

In the procedure `covariance` the lower diagonal matrix of the covariance matrix, R , is determined from the (speech) data array, y . The relation (5.10) is used for $j = 0$ and $i = 0$, while for $1 \leq i \leq M$ factorization is applied. For the values with $j \neq 0$ the recursive relation (5.11) is used. For an *accurate* working procedure covariance the order of the model, $M1$, must be smaller or equal than the amount of data, $N1$, divided by three, or

$$M1 \leq \frac{N1}{3}.$$

```

PROCEDURE covariance ( var y : data; var R : covar; M1 : integer ) ;
(* WARNING ----- ONLY FOR N1 >= 3*M1 !!!! ----- *)
var i, j, k, imod, nterm,
    p, pl, pr, pstrt, pstop : integer ;
    rr : double ;
begin
  rr := 0 ;
  for i := M1 to N - 1 do rr := rr + y[i] * y[i] ;
  R[0,0] := rr ;
  for i := 1 to M1 do
    begin
      imod := 2 * i ;
      nterm := N - 1 - i ;
      rr := 0 ;
      for j := M to M1 + i - 1 do
        begin
          pr := j - i ;
          p := j ;
          repeat
            pl := pr ;
            pr := p + i ;
            rr := rr + y[p] * ( y[pl] + y[pr] ) ;
            p := p + imod ;
          until p > nterm ;
        end ;
      if pr <> N - 1 then

```



```

begin
  if (N - 1 - pr) < i then
    begin
      pstrt := pl + i + 1 ;
      pstop := nterm ;
    end
  else
    begin
      pstrt := N - imod ;
      pstop := pr ;
    end ;
  for p := pstrt to pstop do rr := rr + y[p] * y[p+i] ;
end ;
R[i,0] := rr ;
end;
for i := 1 to M1 do R[i,1] := R[i-1,0] - y[N-i]*y[N-1]
                        + y[M1-i]*y[M1-1] ;

for i := 2 to M1 do
for j := 2 to i do R[i,j] := R[i-1,j-1] - y[N-i]*y[N-j]
                        + y[M1-i]*y[M1-j] ;

end ; { end of covariance }

```

The procedure **Cholesky1** determines the prediction coefficients in the array, a, from the covariance matrix, R, obtain by the previous procedure. Here the first Cholesky method is used. The procedure **Cholesky2** follows the second method, given in 5.16, and gives also the (generalized) reflection coefficients in array, rc. The energy of the residual signal is given in rc[0].

```

PROCEDURE Cholesky1(var R : covar; var a : autocor; M1 : integer ) ;
var i, j, k : integer ;
    sum : single ;
    my : array[1..Mmax] of single ;
    L : array[1..Mmax,1..Mmax] of single ;
begin
  L[1,1] := sqrt(R[1,1]) ;                      {begin first step}
  for i := 2 to M1 do L[i,1] := R[i,1]/L[1,1] ;
  for i := 2 to M1 do
    begin
      for j := 2 to i-1 do
        begin
          sum := 0 ;
          for k := 1 to j-1 do sum := sum + L[i,k]*L[j,k] ;
          L[i,j] := (R[i,j]-sum)/L[j,j] ;
        end ;
        sum := 0 ;
        for k := 1 to i-1 do sum := sum+sqr(L[i,k]) ;
        L[i,i] := sqrt(R[i,i]-sum) ;
      end ;                      {end first step}
    end ;
  end ;

```

```

for i := 1 to M1 do                                {begin second step}
begin
  sum := 0 ;
  for j := 1 to i - 1 do sum := sum + my[j] * L[i,j] ;
  my[i] := -( R[i,0] + sum ) / L[i,i] ;
end ;                                              {end second step}
for i := M1 downto 1 do                            {begin third step}
begin
  sum := 0 ;
  for j := i+1 to M1 do sum := sum + L[j,i]*a[j] ;
  a[i] := ( my[i] - sum ) / L[i,i] ;
end ;
a[0] := 1 ;                                       {end third step}
end ; { end of Cholesky1 }

PROCEDURE Cholesky2 ( var R : covar; var rc, a : autocor; M1 : integer ) ;
var i, j, k : integer ;
    sum : single ;
    L : array[1..Mmax,1..Mmax] of single ;
begin
  for i := 1 to M1 do L[i,1] := R[i,1] ;          {begin first step}
  for i := 2 to M1 do
    for j := 2 to i do
      begin
        sum := 0 ;
        for k := 1 to j - 1 do sum := sum + L[i,k] * L[j,k] / L[k,k] ;
        L[i,j] := R[i,j] - sum ;
      end ;                                       {end first step}
    end ;
    for i := 1 to M1 do                          {begin second step}
      begin
        sum := 0 ;
        for j := 1 to i - 1 do sum := sum + rc[j] * L[i,j] ;
        rc[i] := ( R[i,0] - sum ) / L[i,i] ;
      end ;                                       {end second step}
    end ;
    sum := R[0,0] ;
    for i := 1 to M1 do sum := sum - rc[i] * rc[i] * sum ;
    rc[0] := sum ;                               {calculate energy}
    for i := M1 downto 1 do                      {begin third step}
      begin
        sum := 0 ;
        for j := i+1 to M1 do sum := sum - L[j,i]*a[j] ;
        a[i] := sum/L[i,i] - rc[i];
      end ;
      a[0] := 1 ;                               {end third step}
    end ;
  end ; { end of Cholesky2 }

```

The procedures **Burg** and **Burg2** calculate the reflection coefficients in array, rc, from the (speech) data array, y. The energy of the residual signal is given in rc[0].

```
PROCEDURE Burg ( var y : data; var rc : autocor; N1, M1 : integer ) ;
```

```
var i, p : integer ;
```

```
    energ, temp, nom, den, fakt : single ;
```

```
    e, r : data ;
```

```
begin
```

```
    energ := 0 ;
```

```
    for i := 0 to N1-1 do
```

```
        begin
```

```
            energ := energ + sqr( y[i] ) ;
```

```
            e[i] := y[i] ;
```

```
            r[i] := y[i] ;
```

```
        end ;
```

```
    den := 2 * energ ;
```

```
    fakt := 1 ;
```

```
    for p := 1 to M1 do
```

```
        begin
```

```
            nom := 0 ;
```

```
            den := fakt * den - sqr(e[p-1]) - sqr(r[N1-1]) ;
```

```
            for i := p to N1-1 do
```

```
                begin
```

```
                    nom := nom + e[i] * r[i-1] ;
```

```
                end ;
```

```
            rc[p] := 2 * nom / den ;
```

```
            fakt := 1 - sqr( rc[p] ) ;
```

```
            energ := energ * fakt ;
```

```
            for i := N1-1 downto p do
```

```
                begin
```

```
                    temp := e[i] ;
```

```
                    e[i] := temp - rc[p] * r[i-1] ;
```

```
                    r[i] := r[i-1] - rc[p] * temp ;
```

```
                end ;
```

```
            end ;
```

```
    rc[0] := energ ;
```

```
end ; { end of Burg }
```

```
PROCEDURE Burg2 ( var x : data; var rc : autocor; N1, M1 : integer ) ;
```

```
(* WARNING ----- ONLY FOR M1 is EVEN !!!! ----- *)
```

```
var i, p, ms : integer ;
```

```
    d_0, d_1, d_2, n_0, n_1, n_2 : single ;
```

```
    h1, h2, h3, h4, h5 : single ;
```

```
    s : array[0..5] of single ;
```

```
    energ, n, d, temp : single ;
```

```
    e, r : data ;
```

```
    b : autocor ;
```

```

    cr, ci : glnp ;
begin
    energ := 0 ;
    for i := 0 to N1-1 do
    begin
        e[i] := x[i] ;
        r[i] := x[i] ;
        energ := energ + sqr( x[i] ) ;
    end ;
    p := 2 ;
    while p <= M1 do
    begin
        d_0 := 0 ; d_1 := 0 ; d_2 := 0 ;
        n_0 := 0 ; n_1 := 0 ; n_2 := 0 ;
        for i := p to N1-1 do
        begin
            d_0 := d_0 + sqr(e[i]) + sqr(r[i-2]) ;
            d_1 := d_1 + e[i] * r[i-1] + e[i-1] * r[i-2] ;
            d_2 := d_2 + sqr(e[i-1]) + sqr(r[i-1]) ;
            n_0 := n_0 + e[i] * r[i-2] ;
        end ;
        if p = 2 then
        begin
            rc[p-1] := d_1 / d_2 ;
            d := d_0 - 2*d_1*rc[p-1] + d_2*sqr(rc[p-1]) ;
            n := 2*n_0 - 2*d_1*rc[p-1] + d_2*sqr(rc[p-1]) ;
            rc[p] := n / d ;
        end
        else
        begin
            for i := p to N1-1 do
            begin
                n_1 := n_1 + e[i] * e[i-1] + r[i-1] * r[i-2] ;
                n_2 := n_2 + e[i-1] * r[i-1] ;
            end ;
            d_1 := -2 * d_1 ;
            n_0 := 2 * n_0 ;
            n_1 := -2 * n_1 ;
            n_2 := 2 * n_2 ;
            h1 := sqr(d_0) + sqr( n_0 ) ;
            h2 := sqr(d_1) - sqr(n_1) ;
            h3 := d_1*d_2 - n_1*n_2 ;
            h4 := n_1*d_2 - n_2*d_1 ;
            h5 := sqr(d_2) - sqr(n_2) ;
            s[0] := d_1*h1 - 2*d_0*n_0*n_1 ;
            s[1] := 2*d_0*h2 + 2*d_2*h1 - 4*d_0*n_0*n_2 ;
            s[2] := d_1*(sqr(d_1)-sqr(n_1)) + 6*d_0*h3 + 2*n_0*h4 ;
        end
    end
end

```

```

s[3] := 4*d_1*h3 + 4*d_0*h5 ;
s[4] := d_1*(5*sqr(d_2)-3*sqr(n_2)) - 2*d_2*n_1*n_2 ;
s[5] := 2*d_2*h5 ;
ms := 5 ;
for i := 0 to ms do b[i] := s[ms-i] / s[5] ;
Roots_com (b, cr, ci, ms ) ;
rc[p-1] := cr[5] ;
d := d_0 + d_1*rc[p-1] + d_2*sqr(rc[p-1]) ;
n := n_0 + n_1*rc[p-1] + n_2*sqr(rc[p-1]) ;
rc[p] := n / d ;
end ;
energ := energ * (1-sqr(rc[p-1])) * ( 1-sqr(rc[p])) ;
if p < M1 then
begin
  for i := N1-1 downto p-1 do
  begin
    temp := e[i] ;
    e[i] := temp - rc[p-1] * r[i-1] ;
    r[i] := r[i-1] - rc[p-1] * temp ;
  end ;
  for i := N1-1 downto p do
  begin
    temp := e[i] ;
    e[i] := temp - rc[p] * r[i-1] ;
    r[i] := r[i-1] - rc[p] * temp ;
  end ;
end ;
p := p+2 ;
end ;
rc[0] := energ ;
end ; { end of Burg2 }

```

The procedure Marple determines from the (speech) data array, y, the reflection coefficients, rc, and the prediction coefficients, a. The energy of the residual signal is given in rc[0].

```

PROCEDURE Marple (var y : data; var rc, a : autocor; N1, M1 : integer ) ;
var i, p : integer ;
  help1, help2, help3, help4 : single ;
  epp, rpnm1 : single ;
  alpha, alpha2, alpha3,
  beta1, beta2, beta3,
  gamma1, gamma2, gamma3 : single ;
  epri, epsilon, gamma : single ;
  energ : double ;
  g, w, h, s, v, u, den : single ;
  c, d, cdoubpri, ddoubpri, apri : array[0..Mmax] of single ;
  Rmatr : array[0..Mmax] of single ;
begin

```

```

a[0] := 1 ;                                { initialize for p = 0 }
energ := 0 ;
for i := 0 to N1-1 do energ := energ + sqr(y[i]) ;
energ := 2 * energ ;
c[0] := y[0] / energ ;
d[0] := y[N1-1] / energ ;
g := c[0] * y[0] ;
w := d[0] * y[N1-1] ;
h := c[0] * y[N1-1] ;
s := h ;
v := g ;
u := w ;
den := 1 - g - w ;
epri := energ * den ;
cdoubpri[0] := y[0] / epri ;
ddoubpri[0] := y[N1-1] / epri ;
                                { initialize for p = 1 }
help1 := 0 ;
for i := 1 to N1-1 do help1 := help1 + y[i] * y[i-1] ;
Rmatr[0] := 2 * help1 ;
a[1] := - Rmatr[0] / epri ;
rc[1] := -a[1] ;
energ := (1 - sqr(rc[1])) * epri ;
for p := 1 to M1-1 do
begin
                                { prediction filter update }
epp := y[p] ;
for i := 1 to p do epp := epp + a[i] * y[p-i] ;
rpnm1 := y[N1-1-p] ;
for i := 1 to p do rpnm1 := rpnm1 + a[i] * y[N1-1-p+i] ;

                                { auxiliary vector update }
alpha2 := epp / energ ;
alpha3 := rpnm1 / energ ;
c[0] := alpha2 ;
d[0] := alpha3 ;
for i := 1 to p do
begin
    c[i] := cdoubpri[i-1] + alpha2 * a[i] ;
    d[i] := ddoubpri[i-1] + alpha3 * a[i] ;
end ;

                                { scalar update }
help1 := sqr(epp) / energ ;
help2 := sqr(v) * (1 - w) ;
help3 := sqr(s) * (1 - g) ;
help4 := 2 * s * h * v ;
g := g + help1 + (help2 + help3 + help4) / den ;

```

```

help1 := sqr( rpnm1 ) / energ ;
help2 := sqr(s) * (1 - w) ;
help3 := sqr(u) * (1 - g) ;
help4 := 2 * s * h * u ;
w := w + help1 + (help2 + help3 + help4) / den ;
h := 0 ;
s := 0 ;
v := 0 ;
u := 0 ;
for i := 0 to p do
begin
  h := h + y[Nl-1-p+i] * c[i] ;
  s := s + y[Nl-1-i] * c[i] ;
  v := v + y[i] * c[i] ;
  u := u + y[Nl-1-i] * d[i] ;
end ;

{ denominator update }
den := (1 - w) * (1 - g) - sqr(h) ;
{ time shift update }

help1 := sqr(epp) * (1 - w) ;
help2 := sqr(rpnm1) * (1 - g) ;
help3 := 2 * h * epp * rpnm1 ;
alpha := 1 / (1 + (help1 + help2 + help3) / (energ * den)) ;
epri := alpha * energ ;
beta1 := (h * rpnm1 + epp * (1 - w)) / den ;
gamma1 := (rpnm1 * (1 - g) + h * epp) / den ;
beta2 := (s * h + v * (1 - w)) / den ;
beta3 := (u * h + s * (1 - w)) / den ;
gamma2 := (v * h + s * (1 - g)) / den ;
gamma3 := (s * h + u * (1 - g)) / den ;
for i := 0 to p do
begin
  apri[i] := alpha * (a[i] + beta1 * c[i] + gamma1 * d[i]) ;
  cdoubpri[i] := c[i] + beta2 * c[p-i] + gamma2 * d[p-i] ;
  ddoubpri[i] := d[i] + beta3 * c[p-i] + gamma3 * d[p-i] ;
end ;

{ order update }
for i := p downto 1 do
  Rmatr[i] := Rmatr[i-1] - y[p] * y[i-1] - y[Nl-1-p] * y[Nl-i] ;
help1 := 0 ;
for i := p+1 to Nl-1 do help1 := help1 + y[i-p-1] * y[i] ;
Rmatr[0] := 2 * help1 ;
epsilon := 0 ;
for i := 0 to p do epsilon := epsilon + apri[i] * Rmatr[i] ;
gamma := epsilon / epri ;
rc[p+1] := gamma ;
energ := (1 - sqr(gamma)) * epri ;

```

```

    for i := 1 to p do a[i] := apri[i] - gamma * apri[p+1-i] ;
    a[p+1] := - gamma ;
end ;
rc[0] := energ/2 ;
end ; { end of Marple }

```

The procedure Morf determines from the (speech) data array, y, the forward reflection coefficients, rcf, the backward reflection coefficients, rcb, the forward prediction coefficients, a, and the backward prediction coefficient, b. The energy of the forward residual signal is given in rcf[0] and rcb[0] contains the energy of the backward residual signal.

```

PROCEDURE Morf (var y : data; var rcf, rcb, a, b : autocor; N1, M1 : integer ) ;
var i,p : integer ;
    help : single ;
    epp, rpnmi : single ;
    alphapr, alphadpr, alpha2, alpha3,
    gamma, beta : single ;
    epri, rpri, epsilonp, epsilonm : single ;
    fenerg, benerg : double ;
    g, w, h : single ;
    apri, bpri, c, d, cpri, dpri : array[0..Mmax] of single ;
    Rmatr, R0matr : array[0..Mmax] of single ;
begin
    fenerg := 0 ; { initialize for p = 0 }
    for i := 0 to N1-1 do fenerg := fenerg + sqr(y[i]) ;
    benerg := fenerg ;
    a[0] := 1 ;
    c[0] := y[0] / fenerg ;
    d[0] := y[N1-1] / benerg ;
    g := c[0] * y[0] ;
    w := d[0] * y[N1-1] ;
    h := d[0] * y[0] ;
    epri := fenerg - sqr(y[0]) ;
    rpri := benerg - sqr(y[N1-1]) ;
    cpri[0] := c[0] + h * d[0] / (1-w) ;
    dpri[0] := d[0] + h * c[0] / (1-g) ;
    { initialize for p = 1 }
    help := 0 ;
    for i := 1 to N1-1 do help := help + y[i] * y[i-1] ;
    Rmatr[0] := help ;
    R0matr[0] := help ;
    rcf[1] := Rmatr[0] / rpri ;
    rcb[1] := Rmatr[0] / epri ;
    a[1] := - rcf[1] ;
    b[0] := - rcb[1] ; b[1] := 1 ;
    fenerg := epri - rcf[1] * Rmatr[0] ;
    benerg := rpri - rcb[1] * Rmatr[0] ;
    for p := 1 to M1-1 do

```



```

begin
                                { prediction filter update }
    epp := y[p] ;
    for i := 1 to p do epp := epp + a[i] * y[p-i] ;
    rpnm1 := y[N1-1-p] ;
    for i := 1 to p do rpnm1 := rpnm1 + b[p-i] * y[N1-1-p+i] ;
                                { auxiliary vector update }

    alpha2 := epp / fenerg ;
    alpha3 := rpnm1 / benerg ;
    c[0] := alpha2 ;
    d[p] := alpha3 ;
    for i := 1 to p do c[i] := cpri[i-1] + alpha2 * a[i] ;
    for i := 0 to p-1 do d[i] := dpri[i] + alpha3 * b[i] ;
                                { scalar update }
    g := g + sqr(epp) / fenerg + sqr(h) / (1-w) ;
    w := w + sqr(rpnm1) / benerg + sqr(h) / (1-g) ;
    h := 0 ;
    for i := 0 to p do h := h + y[N1-1-i] * c[i] ;
                                { time shift update }
    alphapr := 1 / (1 + sqr(epp) / (fenerg * (1-g))) ;
    alphadpr := 1 / (1 + sqr(rpnm1) / (benerg * (1-w))) ;
    for i := 0 to p do
begin
    apri[i] := alphapr * ( a[i] + epp * c[i] / (1-g)) ;
    bpri[i] := alphadpr * ( b[i] + rpnm1 * d[i] / (1-w)) ;
    cpri[i] := c[i] + h * d[i] / (1-w) ;
    dpri[i] := d[i] + h * c[i] / (1-g) ;
end ;
    epri := alphapr * fenerg ;
    rpri := alphadpr * benerg ;
                                { aorder update }

    for i := p downto 1 do
begin
    Rmatr[i] := Rmatr[i-1] - y[N1-1-p] * y[N1-i] ;
    ROMatr[i-1] := ROMatr[i-1] - y[p] * y[i-1] ;
end ;
    help := 0 ;
    for i := p+1 to N1-1 do help := help + y[i-p-1] * y[i] ;
    Rmatr[0] := help ;
    ROMatr[p] := help ;
    epsilonp := 0 ;
    epsilonm := 0 ;
    for i := 0 to p do
begin
    epsilonp := epsilonp + apri[i] * Rmatr[i] ;
    epsilonm := epsilonm + bpri[p-i] * ROMatr[p-i] ;
end ;

```

```

gamma := epsilonp / rpri ;
beta  := epsilonm / epri ;
rcf[p+1] := gamma ;
rcb[p+1] := beta ;
fenerg := epri - gamma * epsilonm ;
benerg := rpri - beta * epsilonp ;
for i := 1 to p do
begin
  a[i] := apri[i] - gamma * bpri[i-1] ;
  b[i] := bpri[i-1] - beta * apri[i] ;
end ;
a[p+1] := - gamma ;
b[0] := - beta ;
b[p+1] := 1 ;
end ;
rcf[0] := fenerg ;
rcb[0] := benerg ;
end ; { ene of Morf }

```

Bibliography

- [1] Bell, B.M. and D.B. Percival
A TWO STEP BURG ALGORITHM.
IEEE Trans. on Signal Processing, Vol. SP-39(1991), p. 185-189.
- [2] Chen, C.H.
SIGNAL PROCESSING HANDBOOK.
New York: Dekker, 1988.
- [3] Choi, B.S. and T.M. Cover
AN INFORMATION-THEORETIC PROOF OF BURG'S MAXIMUM ENTROPY SPECTRUM.
Proceedings of IEEE, Vol. 72(1984), p. 1094-1095.
- [4] Delsarte, P. and Y. Genin, Y. Kamp, P. van Dooren
SPEECH MODELLING AND THE TRIGONOMETRIC MOMENT PROBLEM.
Philips J. Res., Vol. 37(1982), p. 277-292.
- [5] Delsarte, P. and Y.V. Genin
THE SPLIT LEVINSON ALGORITHM.
IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-34(1986), p. 470-478.
- [6] Goldberg, A.J. and H.L. Shaffer
A REAL-TIME ADAPTIVE PREDICTIVE CODER USING SMALL COMPUTERS.
IEEE Trans. on Communication, Vol. COM-23(1975), p. 1443-1451.
- [7] Furui, S.
DIGITAL SPEECH PROCESSING, SYNTHESIS AND RECOGNITION.
New York: Dekker, 1989.
- [8] Furui, S. and M.H. Sondhi
ADVANCES IN SPEECH SIGNAL PROCESSING.
New York: Dekker, 1991.
- [9] Jayant, N.S. and P. Noll
DIGITAL CODING OF WAVEFORMS.
Englewood Cliffs, NJ: Prentice-Hall, 1984.
- [10] Kabal, P. and R.P. Ramachandran
THE COMPUTATION OF LINE SPECTRAL FREQUENCIES USING CHEBYSHEV POLYNOMIALS.
IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-34(1986), p. 1419-1425.

- [11] Kailath, T.
SIGNAL PROCESSING IN THE VLSI ERA.
In: *VLSI AND MODERN SIGNAL PROCESSING*. Ed. by S.Y. Kung and H.J. Whitehouse, T. Kailath. Englewood Cliffs, NJ: Prentice-Hall, 1985. P. 5-24.
- [12] Kailath, T.
LINEAR ESTIMATION FOR STATIONARY AND NEAR-STATIONARY PROCESSES.
In: *MODERN SIGNAL PROCESSING*. Ed. by T. Kailath. New York: Springer, 1985. P. 59-128.
- [13] Makhoul, J.
SPEECH CODING AND PROCESSING.
In: *MODERN SIGNAL PROCESSING*. Ed. by T. Kailath. New York: Springer, 1985. P. 211-247.
- [14] Markel, J.D. and A.H. Gray
LINEAR PREDICTION OF SPEECH.
New York: Springer, 1976.
- [15] Marple Jr, S.L.
A NEW AUTOREGRESSIVE SPECTRUM ANALYSIS ALGORITHM.
IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-28(1980), p. 441-454.
- [16] Marple Jr, S.L.
DIGITAL SPECTRAL ANALYSIS WITH APPLICATIONS.
Englewood Cliffs, NJ: Prentice-Hall, 1987.
- [17] Morf, M. and B. Dickinson, T. Kailath, A. Vieira
EFFICIENT SOLUTION FOR COVARIANCE EQUATIONS FOR LINEAR PREDICTION.
IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-25(1977), p. 429-433.
- [18] Papamichalis, P.E.
PRACTICAL APPROACHES TO SPEECH CODING.
Englewood Cliffs, NJ: Prentice-Hall, 1987.
- [19] Papoulis, A.
LEVINSON ALGORITHM, WOLD'S DECOMPOSITION, AND SPECTRAL ESTIMATION.
SIAM Rev, Vol. 27(1985), p. 405-441.
- [20] Pizer, S.M.
NUMERICAL COMPUTING AND MATHEMATICAL ANALYSIS.
Chicago: Science Research Associates Inc, 1975.
- [21] Press, W.H. and B.P. Flannery, S.A. Teukolsky, W.T. Vetterling
NUMERICAL RECIPES, The art of scientific computing.
Cambridge: Cambridge University Press, 1986.
- [22] Rabiner, L.R. and R.W. Schafer
DIGITAL PROCESSING OF SPEECH SIGNALS.
Englewood Cliffs, NJ: Prentice-Hall, 1978.

- [23] Rivlin, T.J.
THE CHEBYSHEV POLYNOMIALS.
New York: Wiley, 1974.
- [24] Saito, S. and K. Nakata
FUNDAMENTALS OF SPEECH PROCESSING.
New York: Academic, 1985.
- [25] Soong, F.K. and B-H. Juang
LINE SPECTRUM PAIR (LSP) AND SPEECH DATA COMPRESSION.
Proc. IEEE Intern. Conf. Acoust., Speech, Signal Processing, San Diego, Ca., 19-21 March 1984. Vol. 1.
New York: IEEE Publishing Services, 1984.
- [26] Stoer, J. and R. Bulirsch
INTRODUCTION TO NUMERICAL ANALYSIS.
New York: Springer, 1980.
- [27] Tremain, T.E.
THE GOVERNMENT STANDARD LINEAR PREDICTION ALGORITHM: LPC-10.
Speech Technology, Vol. 2(1982), p. 40-49.
- [28] Wakita, K.
LINEAR PREDICTION VOICE SYNTHESIZERS: Line-Spectrum Pairs (LSP) is the newest of several techniques.
Speech Technology, Vol. 1(1981), p. 17-22.

Index

ADPCM, vi, 4
algorithm, vi, vii, 2, 4, 8–10, 27, 28, 30–33, 41–43, 45, 46
APC, vi, 4
autocorrelation, vii, 1, 6, 7, 14, 26–28, 60, 62, 65
Burg, vii, 27, 31–33, 42, 75
Chebyshev, 18, 20
Cholesky, vii, 28, 30, 57
coefficient
 parcor, vi, 8, 56
 reflection, vi, 8, 10, 14, 17, 23, 28–30, 60, 62, 64, 65, 73, 75, 77, 80
covariance, vii, 26, 27, 29, 30, 57
DPCM, 3, 4
Durbin, vi, vii, 7, 8, 27, 28, 30, 31, 38, 51
filter, vii, 1–4, 11–13
 analysis, vi, 1–3, 11, 13, 17, 18, 65
 inverse, 2
 lattice, vi, 4, 6, 7, 10–13, 19, 65
 synthesis, vi, 1–5, 11, 12, 17, 18, 21, 65
 transversal, vi, 4, 10, 12
Levinson, vi, vii, 7–9, 14–17, 23, 27, 28, 30, 31, 38, 51, 62
LPC, vi, 4
LSE, vii, 26, 27
LSP, vi, vii, 17, 18, 22
Marple, 27, 33, 41–43, 45, 77
matrix, 7, 8, 12, 13, 17, 27–30, 34–36, 42, 51–55, 57, 59
 autocorrelation, 26, 28, 51
 companion, 22
 covariance, 29, 42, 72, 73
 data, 29, 34
 equation, 7, 8, 28, 30, 34, 57–59

Hessenberg, 22
 triangular, 28, 30, 57–59
model
 AR, vi, 2, 5
 ARMA, 2
 MA, 2
 signal, vi, 1, 3, 4
Morf, vii, 46, 80
MSE, 6, 7, 26
polynomial, 8, 9, 15, 17–25, 30, 32, 33, 54, 60, 66, 67, 71
prediction, vi, vii, 1–9, 11, 26, 30, 42, 53, 55, 56
 coefficient, 3, 4, 6–8, 10, 23, 27, 28, 30, 32–34, 42, 45, 46, 60, 62, 64, 66, 73, 77, 80
 error, 2, 3, 6, 7, 11, 12, 26, 27, 31, 34, 42, 55, 56, 62
 order, 6, 9, 12, 17, 26, 28, 30, 34, 60
Schur, vi, vii, 8, 14–16, 23, 65
speech, vi, vii, 1, 2, 4, 26, 30, 60, 72, 75, 77, 80
Toeplitz, vii, 7, 26–28, 34, 51, 52, 54
Walker, vii, 7, 28, 30, 54, 55
window, vii, 26–29, 31
Yule, vii, 7, 28, 30, 54, 55

- (236) Lanners, J.O.
KNOWLEDGE BASED ADAPTIVE BLOOD PRESSURE CONTROL: A Simplexys expert system application.
EUT Report 90-E-236. 1990. ISBN 90-6144-236-2
- (237) Ren Qingchang
PREDICTION ERROR METHOD FOR IDENTIFICATION OF A HEAT EXCHANGER.
EUT Report 90-E-237. 1990. ISBN 90-6144-237-0
- (238) Lanners, J.O.
THE USE OF PETRI NET THEORY FOR SIMPLEXYS EXPERT SYSTEMS PROTOCOL CHECKING.
EUT Report 90-E-238. 1990. ISBN 90-6144-238-9
- (239) Wang, X.
PRELIMINARY INVESTIGATIONS ON TACTILE PERCEPTION OF GRAPHICAL PATTERNS.
EUT Report 90-E-239. 1990. ISBN 90-6144-239-7
- (240) Lutgens, J.M.A.
KNOWLEDGE BASE CORRECTNESS CHECKING FOR SIMPLEXYS EXPERT SYSTEMS.
EUT Report 90-E-240. 1990. ISBN 90-6144-240-0
- (241) Brinker, A.C. den
A MEMBRANE MODEL FOR SPATIOTEMPORAL COUPLING.
EUT Report 90-E-241. 1990. ISBN 90-6144-241-9
- (242) Kwaspen, J.J.M. and H.C. Heyker, J.I.M. Demarteau, Th.G. van de Roer
MICROWAVE NOISE MEASUREMENTS ON DOUBLE BARRIER RESONANT TUNNELING DIODES.
EUT Report 90-E-242. 1990. ISBN 90-6144-242-7
- (243) Massee, P. and H.A.L.M. de Graaf, W.J.M. Balemans, H.G. Knoopers, H.H.J. ten Kate
PREDESIGN OF AN EXPERIMENTAL (5-10 MWt) DISK MHD FACILITY AND PROSPECTS OF COMMERCIAL (1000 MWt) MHD/STEAM SYSTEMS.
EUT Report 90-E-243. 1990. ISBN 90-6144-243-5
- (244) Klompstra, Martin and Ton van den Boom, Ad Damen
A COMPARISON OF CLASSICAL AND MODERN CONTROLLER DESIGN: A case study.
EUT Report 90-E-244. 1990. ISBN 90-6144-244-3
- (245) Berg, P.H.G. van de
ON THE ACCURACY OF RADIOWAVE PROPAGATION MEASUREMENTS: Olympus propagation experiment.
EUT Report 90-E-245. 1990. ISBN 90-6144-245-1
- (246) Maaqt, P.J.I. de
A SYNTHESIS METHOD FOR COMBINED OPTIMIZATION OF MULTIPLE ANTENNA PARAMETERS AND ANTENNA PATTERN STRUCTURE.
EUT Report 90-E-246. 1990. ISBN 90-6144-246-X
- (247) Józwiak, L. and T. Spassova-Kwaaitaal
DECOMPOSITIONAL STATE ASSIGNMENT WITH REUSE OF STANDARD DESIGNS: Using counters as sub-machines and using the method of maximal adjacencies to select the state chains and the state codes.
EUT Report 90-E-247. 1990. ISBN 90-6144-247-8
- (248) Hoeijmakers, M.J. and J.M. Vleeshouwers
DERIVATION AND VERIFICATION OF A MODEL OF THE SYNCHRONOUS MACHINE WITH RECTIFIER WITH TWO DAMPER WINDINGS ON THE DIRECT AXIS.
EUT Report 90-E-248. 1990. ISBN 90-6144-248-6

- (249) Zhu, Y.C. and A.C.P.M. Backx, P. Eykhoff
MULTIVARIABLE PROCESS IDENTIFICATION FOR ROBUST CONTROL.
EUT Report 91-E-249. 1991. ISBN 90-6144-249-4
- (250) Pfaffenhöfer, F.M. and P.J.M. Cluitmans, H.M. Kuipers
EMDABS: Design and formal specification of a datamodel for a clinical research database system.
EUT Report 91-E-250. 1991. ISBN 90-6144-250-8
- (251) Eindhoven, J.T.J. van and G.G. de Jong, L. Stok
THE ASCIS DATA FLOW GRAPH: Semantics and textual format.
EUT Report 91-E-251. 1991. ISBN 90-6144-251-6
- (252) Chen, J. and P.J.I. de Maagt, M.H.A.J. Herben
WIDE-ANGLE RADIATION PATTERN CALCULATION OF PARABOLOIDAL REFLECTOR ANTENNAS: A comparative study.
EUT Report 91-E-252. 1991. ISBN 90-6144-252-4
- (253) Haan, S.W.H. de
A PWM CURRENT-SOURCE INVERTER FOR INTERCONNECTION BETWEEN A PHOTOVOLTAIC ARRAY AND THE UTILITY LINE.
EUT Report 91-E-253. 1991. ISBN 90-6144-253-2
- (254) Velde, M. van de and P.J.M. Cluitmans
EEG ANALYSIS FOR MONITORING OF ANESTHETIC DEPTH.
EUT Report 91-E-254. 1991. ISBN 90-6144-254-0
- (255) Smolders, A.B.
AN EFFICIENT METHOD FOR ANALYZING MICROSTRIP ANTENNAS WITH A DIELECTRIC COVER USING A SPECTRAL DOMAIN MOMENT METHOD.
EUT Report 91-E-255. 1991. ISBN 90-6144-255-9
- (256) Backx, A.C.P.M. and Damen, A.A.H.
IDENTIFICATION FOR THE CONTROL OF MIMO INDUSTRIAL PROCESSES.
EUT Report 91-E-256. 1991. ISBN 90-6144-256-7
- (257) Maagt, P.J.I. de and H.G. ter Morsche, J.L.M. van den Broek
A SPATIAL RECONSTRUCTION TECHNIQUE APPLICABLE TO MICROWAVE RADIOMETRY
EUT Report 92-E-257. 1992. ISBN 90-6144-257-5
- (258) Vleeshouwers, J.M.
DERIVATION OF A MODEL OF THE EXCITER OF A BRUSHLESS SYNCHRONOUS MACHINE.
EUT Report 92-E-258. 1992. ISBN 90-6144-258-3
- (259) Orlov, V.B.
DEFECT MOTION AS THE ORIGIN OF THE 1/F CONDUCTANCE NOISE IN SOLIDS.
EUT Report 92-E-259. 1992. ISBN 90-6144-259-1
- (260) Rooijackers, J.E.
ALGORITHMS FOR SPEECH CODING SYSTEMS BASED ON LINEAR PREDICTION.
EUT Report 92-E-260. 1992. ISBN 90-6144-260-5