



LUND UNIVERSITY

Polynomial Factorization Using Recursive Formulas for Complex Integrals

Zhou, Zhao-Ying; Åström, Karl Johan

1981

Document Version:

Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for published version (APA):

Zhou, Z.-Y., & Åström, K. J. (1981). *Polynomial Factorization Using Recursive Formulas for Complex Integrals*. (Technical Reports TFRT-7223). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:

2

General rights

Unless other specific re-use rights are stated the following general rights apply:

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

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

CODEN: LUTFD2/(TFRT-7223)/1-030/(1981)

POLYNOMIAL FACTORIZATION USING RECURSIVE FORMULAS
FOR COMPLEX INTEGRALS

ZHOU ZHAO-YING

KARL JOHAN ÅSTRÖM

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
JUNE 1981

POLYNOMIAL FACTORIZATION
USING RECURSIVE FORMULAS FOR COMPLEX INTEGRALS

Zhou Zhao-ying
Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology

April 1981

LUND INSTITUTE OF TECHNOLOGY DEPARTMENT OF AUTOMATIC CONTROL Box 725 S 220 07 Lund 7 Sweden		Document name	
		Date of issue June 1981	
Author(s) Zhou Zhao-ying Karl Johan Åström		Document number CODEN: LUTFD2/(TFRT-7223)/1-030/(1981)	
		Supervisor	
Title and subtitle Polynomial factorization using recursive formulas for complex integrals		Sponsoring organization	
Abstract			
<p>The complex integrals</p> $I(\lambda) = \frac{1}{2\pi i} \int_{-\lambda-1}^{\lambda+1} \frac{\rho A(z)A(z^{-1}) + B(z)B(z^{-1})}{C(z)C(z^{-1})} \frac{dz}{z}$ <p>can be used to solve the spectral factorization problem</p> $P(z)P(z^{-1}) = \rho A(z)A(z^{-1}) + B(z)B(z^{-1})$ <p>which appears in LQG calculations. This method has high precision.</p> <p>The report gives recursive formulas for evaluating $I(\lambda)$ and two iterative algorithms for the spectral factorization. Two test programs are written in PASCAL. The calculation of $I(0)$ also gives a test for the stability of the polynomial $P(z)$ in each iteration.</p>			
Key words			
Classification system and/or index terms (if any)			
Supplementary bibliographical information			
ISSN and key title		ISBN	
Language English		Recipient's notes	
Security classification		Number of pages 30	

CONTENT

1. INTRODUCTION
 2. ITERATION METHOD AND RELATIONSHIPS
 3. RECULSIVE FORMULAS FOR COMPLEX INTEGRALS I(1)
 4. REDUCED VERSION
 5. REFERENCES
- APPENDIX A: PROGRAM LISTING REFAC1
- B: PROGRAM LISTING REFAC2

1. INTRODUCTION

Consider a discrete time system governed by the model

$$A(z) y(t) = B(z) u(t) \quad (1.1)$$

Let the criterion be to minimize

$$J = \sum_{t=0}^{\infty} [y^2(t) + \rho u^2(t)] \quad (1.2)$$

The solution of this optimization problem gives a closed loop system whose characteristic polynomial $P(z)$ is given by

$$P(z)P(z^{-1}) = \rho A(z)A(z^{-1}) + B(z)B(z^{-1}) \quad (1.3)$$

where P , A and B are polynomials with real coefficients.

$$P(z) = p_0 z^k + p_1 z^{k-1} + \dots + p_k$$

$$A(z) = a_0 z^k + a_1 z^{k-1} + \dots + a_k$$

$$B(z) = b_0 z^k + b_1 z^{k-1} + \dots + b_k$$

and ρ is a weighting factor. See Astrom (1979). For self-tuning regulators it is useful to have efficient algorithms to solve the design problem.

It can be shown that the right hand side of (1.3) can be factorized when $\rho \geq 0$. Consider a stationary discrete stochastic process

$$y(t) = r A(z) w(t) + B(z) v(t) \quad (1.4)$$

where $w(t)$ and $v(t)$ are uncorrelated white noises and the coefficients a_i and b_i are real numbers. Consider also another stationary discrete time process with white noise input $\epsilon(t)$

$$z(t) = P(z) \epsilon(t) \quad (1.5)$$

where $P(z)$ is a real coefficient polynomial. These two processes are equivalent in the sense of that they have same spectral density function, i.e.

$$\phi_y(\omega) = \phi_z(\omega)$$

Let $\rho = r^2$, then (1.3) follows. (1.3) has several solutions because of

$$\begin{aligned} P(z)P(z^{-1}) &= k_1 \prod_{i=1}^k (z - r_i)(z^{-1} - r_i^{-1}) \\ &= k_2 \prod_{i=1}^k (z - 1/r_i)(z^{-1} - 1/r_i) \end{aligned} \quad (1.6)$$

where k_1 and k_2 are constants. The resultant polynomial $P(z)$ is the combination of all possible zeros, but there is only one combination with

$$|z_i| \leq 1 \quad \text{for } i=0,1,\dots,k$$

which corresponds to a stable polynomial $P(z)$.

The complex integral

$$I(k) = \frac{1}{2\pi i} \oint_C \frac{\rho A(z)A(z^{-1}) + B(z)B(z^{-1})}{C(z)C(z^{-1})} dz \quad (1.7)$$

is related to the factorization problem. The efficient algorithm Astrom et al (1970) and Astrom (1970) for evaluating $I(0)$ can be used to solve (1.7), and the calculation of $I(0)$ can also give a test for the stability of the polynomial $P(z)$, which is described in previous literatures.

This report is organized as follows. The iteration method and the relationship between $I(k)$ and coefficients of $P(z)$ is described in section 2. The recursive formulas are given in section 3. A numerical example shows that the method based on the complex integral has high precision. Section 4 shows a reduced version of recursive formulae which requires less computing times than the former. Two programs in PASCAL are given in appendices.

2. ITERATION_METHOD_AND_RELATIONSHIPS

A method based on Newton iteration is suggested by Kucera et al (1976) and Kucera (1979) because it features monotone and essentially quadratic convergence. Define polynomials $C(z)$ and $X(z)$ with

$$\deg C_J(z) = \deg X_J(z) = \deg P_J(z)$$

where subscript J denotes the J th iteration. The iteration is performed recursively by solving

$$C_J(z)X_J(z^{-1}) + X_J(z)C_J(z^{-1}) = \rho A(z)A(z^{-1}) + B(z)B(z^{-1}) \quad (2.1)$$

starting from a stable polynomial $C_0(z)$. The next polynomial in the iteration is then given by

$$C_{J+1}(z) = -\frac{1}{2} [C_J(z) + X_J(z)] \quad (2.2)$$

and the desired polynomial $P(z)$ is obtained as

$$P(z) = \lim_{J \rightarrow \infty} C_J(z) \quad (2.3)$$

In practice it only takes a few steps for $C(z)$ to converge to the desired polynomial. The test

$$|c_{i,j+1} - c_{i,j}| < \epsilon \quad \text{for } i=0,1,\dots,n$$

where ϵ is a given small value, can be used to stop the recursion.

Equation (2.1) can be written as a set of linear equations

$$CX = 2(\rho a + b) \quad (2.4)$$

where C is a square matrix of dimension k given by

$$C = \begin{bmatrix} 2c_0 & 2c_1 & 2c_2 & \dots & 2c_{k-2} & 2c_{k-1} & 2c_k \\ c_1 & c_0 + c_2 & c_1 + c_3 & \dots & c_{k-3} + c_{k-1} & c_{k-2} + c_k & c_{k-1} \\ c_2 & c_3 & c_0 + c_4 & \dots & c_{k-4} + c_k & c_{k-1} & c_{k-2} \\ \dots & & & & & & \dots \\ c_{k-1} & c_k & 0 & \dots & 0 & c_0 & c_1 \\ c_k & 0 & 0 & \dots & 0 & 0 & c_0 \end{bmatrix}$$

and a and b are vectors of the same form

$$a = \begin{bmatrix} \sum_{i=0}^k a_i \\ \sum_{i=0}^{k-1} a_{i+1} \\ \sum_{i=0}^{k-2} a_{i+2} \\ \dots \\ \sum_{i=0}^1 a_{i+k-1} \\ a_k \\ 0 \end{bmatrix}$$

If $C(z)$ is a stable polynomial, dividing both sides of (2.1) by $C(z)C(z^{-1})$ gives

$$\frac{C(z)X(z^{-1}) + X(z)C(z^{-1})}{C(z)C(z^{-1})} = \frac{2[\varrho A(z)A(z^{-1}) + B(z)B(z^{-1})]}{C(z)C(z^{-1})} \\ = 2 \sum_{\lambda=-\infty}^{\infty} I(\lambda) z^{\lambda} \quad (2.5)$$

$$\text{or } \frac{X(z)}{C(z)} = I(0) + 2 \sum_{\lambda=-\infty}^{-1} I(\lambda) z^{\lambda} \quad (2.6)$$

Applying long division to the left side of (2.6) we have

$$\frac{X(z)}{C(z)} = \frac{x_0}{c_0} + \frac{1}{c_0} \left(x_1 - \frac{c_1}{c_0} x_0 \right) z^{-1} + \dots \quad (2.7)$$

Equating coefficients of same powers of (2.6) and (2.7) gives

$$I(0) = x_0 / c_0$$

$$I(1) = (x_1 - x_0 c_1 / c_0) / 2c_0$$

...

$$I(k) = [x_k - x_{k-1} c_1 / c_0 - (x_{k-2} - x_{k-2} c_1 / c_0) c_{k-1} / c_0 - \dots - 1/2 c_0]$$

...

$$\text{or } X = C_1 I \quad (2.8)$$

where

$$c_1 = \begin{bmatrix} c_0 & & & 0 \\ c_1 & 2c_0 & & \\ \vdots & \vdots & \ddots & \vdots \\ c_k & 2c_{k-1} & \dots & 2c_0 \end{bmatrix}$$

EXAMPLE 1.

Given the polynomials

$$P(z) = p_0 z + p_1 \quad \text{and}$$

$$C(z) = c_0 z + c_1$$

Solve

$$\begin{bmatrix} 2c_0 & 2c_1 \\ c_1 & c_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 2(p_0^2 + p_1^2) \\ 2p_0 p_1 \end{bmatrix}$$

We find

$$x_0 = [(p_0^2 + p_1^2)c_0 - 2p_0 p_1 c_1] / (c_0^2 - c_1^2)$$

$$x_1 = [2p_0 p_1 c_0 - (p_0^2 + p_1^2)c_1] / (c_0^2 - c_1^2)$$

Substituting x_0 and x_1 into (2.7), we obtain

$$\begin{aligned} 2I(1) &= (c_0 x_1 - c_1 x_0) / c_0^2 \\ &= [2p_0 p_1 (c_0^2 + c_1^2) - 2(p_0^2 + p_1^2)c_0 c_1] / [c_0^2 (c_0^2 - c_1^2)] \end{aligned}$$

Integral $I(1)$ can be also obtained by use of residue theorem

Let

$$\phi(z) = \frac{P(z)P(z^{-1})}{C(z)C(z^{-1})} z^{-2}$$

Then

$$\begin{aligned}
 I(1) &= \sum_i \operatorname{Res} \{ \phi(z) \}_{z=z_i} \\
 &= \lim_{z \rightarrow 0} [z^2 \phi(z)]' + \lim_{z \rightarrow -c/c_1} [(z - c/c_1) \phi(z)] \\
 &= [p_0 p_1 (c_1^2 + c_2^2) - (p_0^2 + p_1^2) c c_1] / [c_0^2 (c_0^2 - c_1^2)] \\
 &\quad \square
 \end{aligned}$$

Substituting (2.8) into (2.5) gives a set of linear equations for $I(1)$

$$C C_1 I = \rho a + b \quad (2.9)$$

Notice that the matrix $C C_1$ will be symmetric if its first row is divided by 2.

EXAMPLE 2.

Consider a second order system in the form of (2.9)

$$\begin{bmatrix} (c_0^2 + c_1^2)/2 & c c_1 + c_2 c_0 & c c_2 \\ c c_1 + c_2 c_0 & c_0^2 + c_1^2 & c c_1 \\ c c_2 & c c_0 & c_0^2 \end{bmatrix} \begin{bmatrix} I(0) \\ I(1) \\ I(2) \end{bmatrix} = \begin{bmatrix} (p_0^2 + p_1^2)/2 \\ p_0 p_1 + p_1 p_2 \\ p_0 p_2 \end{bmatrix}$$

A simple nonsingular transformation T_{12} brings $C C_1$ to an upper triangular matrix

$$C C_1 T_{12} = \begin{bmatrix} (c_0^2 + c_1^2)/2 - c_2^2 / (c_0 + c_1) & c c_1 & c c_2 \\ 0 & c_0^2 + c_1^2 & c c_1 \\ 0 & 0 & c_0^2 \end{bmatrix}$$

where

$$T_{12} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -c/c_0 & -c_1/c_0 & 1 \end{bmatrix}$$

and

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ -c_1/(c_1+c_2) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then $I(0)$, $I(1)$ and $I(2)$ can be obtained by back substitution.

□

An alternative approach is to reduce the calculation of $I(1)$ to a problem which involves calculation of square loss function. Recursive equations are given in the next section.

3. RECURSIVE FORMULAS FOR $I_k(\lambda)$

An efficient tool developed by Astrom (1970) for evaluating loss function can be exploited to solve (1.7). Introduce

$$\begin{aligned} C_k^*(z) &= z C_k(z^{-1}) = c_0^k + c_1^k z + \dots + c_k^k z^k \\ B_k^*(z) &= z B_k(z^{-1}) = b_0^k + b_1^k z + \dots + b_k^k z^k \\ \text{and} \quad Q_k(\lambda, z) C_k(z) + R_k(\lambda, z) &= z^{\lambda} B_k(z) \end{aligned} \quad (3.1)$$

where $Q_k(\lambda, z)$ and $R_k(\lambda, z)$ are the quotient and the remainder of the polynomial division (3.1). In this section we only give the formulas for the second term of the right hand side of (2.3) because formulas for the first term have the same form.

Consider the complex integral

$$\begin{aligned} I_k(\lambda) &= \frac{1}{2\pi i} \oint \frac{z^{\lambda} B_k(z) B_k(z^{-1})}{C_k(z) C_k(z^{-1})} dz \\ &= \frac{1}{2\pi i} \oint \left[\frac{Q_k(\lambda, z) B_k^*(z)}{C_k^*(z)} + \frac{R_k(\lambda, z) B_k(z^{-1})}{C_k(z) C_k(z^{-1})} \right] \frac{dz}{z} \\ &= Q_k(\lambda, 0) \frac{b_0}{c_0} + \frac{1}{2\pi i} \oint \frac{R_k(\lambda, z) B_k(z^{-1})}{C_k(z) C_k(z^{-1})} dz \end{aligned} \quad (3.2)$$

Equation (3.1) is then reduced to a simple complex integral. In similar way define

$$\begin{aligned} C_{k-1}(z) &= z^{-1} [C_k(z) - \alpha_k C_k^*(z)] & \alpha_k &= c_k^k / c_0^k \\ B_{k-1}(z) &= z^{-1} [B_k(z) - \beta_k C_k^*(z)] & \beta_k &= b_k^k / c_0^k \\ \text{and } R_{k-1}(\lambda, z) &= z^{-1} [R_k(\lambda, z) - \gamma_k C_k^*(z)] & \gamma_k &= r_k^k(\lambda) / c_0^k \end{aligned}$$

Recursive formulas for the complex integral

$$\begin{aligned}
 I_{k-1}^0(\lambda) &= \frac{1}{2\pi i} \int \frac{R_{k-1}(\lambda, z) B_{k-1}(z^{-1})}{C_{k-1}(z) C_{k-1}(z^{-1})} \frac{dz}{z} \\
 &= \left[\frac{1}{2\pi i} \int \frac{R_{k-1}(\lambda, z) B_{k-1}(z^{-1})}{C_k(z) C_k(z^{-1})} \frac{dz}{z} \right] \frac{1}{1 - \alpha_k}
 \end{aligned}$$

Substituting $R_{k-1}(\lambda, z)$ and $B_{k-1}(z^{-1})$ into the above equality we get

$$\begin{aligned}
 (1 - \alpha_k) I_{k-1}^0(\lambda) &= \frac{1}{2\pi i} \int \frac{[R_k(\lambda, z) - \gamma_k C_k^*(z)] [B_k(z^{-1}) - \beta_k C_k^*(z^{-1})]}{C_k(z) C_k(z^{-1})} \frac{dz}{z} \\
 &= I_k^0(\lambda) - \frac{\beta_k}{2\pi i} \int \frac{R_k(\lambda, z)}{C_k^*(z)} \frac{dz}{z} - \frac{\gamma_k}{2\pi i} \int \frac{B_k(z^{-1})}{C_k^*(z^{-1})} \frac{dz}{z} + \beta_k \gamma_k \\
 &= I_k^0(\lambda) - \beta_k \gamma_k
 \end{aligned}$$

Hence

$$\begin{aligned}
 I_k^0(\lambda) &= (1 - \alpha_k^2) I_{k-1}^0(\lambda) + \beta_k \gamma_k \\
 &= \frac{1}{c_0} \sum_{i=0}^k \frac{b_i^i r_i^i}{c_0} \tag{3.3}
 \end{aligned}$$

and

$$I_k(\lambda) = Q_k(\lambda, 0) \frac{b_0^k}{c_0} + \frac{1}{c_0} \sum_{i=0}^k \frac{b_i^i r_i^i}{c_0} \tag{3.4}$$

Equation (3.4) is a useful recursive formula for computing $I(\lambda)$. The polynomials $Q(\lambda, z)$ and $R(\lambda, z)$ can be obtained in a straightforward manner from (3.1)

$$\left. \begin{aligned}
 Q_k(l, z) &= q_0 z^l + q_1 z^{l-1} + \dots + q_l \\
 R_k(l, z) &= r_1 z^k + r_2 z^{k-1} + \dots + r_k
 \end{aligned} \right\} \tag{3.5}$$

where

$$\left. \begin{aligned}
 q_0 &= b_0 / c_0 \\
 q_1 &= (b_1 - q_0 c_1) / c_0 \\
 &\dots \\
 q_l &= (b_l - q_0 c_l - q_1 c_{l-1} - \dots - q_{l-1} c_1) / c_0 \\
 r_1 &= b_{l+1} - q_0 c_{l+1} - q_1 c_l - \dots - q_l c_1 \\
 r_2 &= b_{l+2} - q_0 c_{l+2} - q_1 c_{l+1} - \dots - q_l c_2 \\
 &\dots \\
 r_k &= -q_0 c_{l+k} - q_1 c_{l+k-1} - \dots - q_l c_k
 \end{aligned} \right\} \tag{3.6}$$

The coefficients of $Q(l, z)$ and $R(l, z)$ can also be obtained by using the following table

$q_0 = b_0/c_0$	0	b_0	b_1	b_2	\dots	b_k
$q_1 = r_1(0)/c_0$	0	c_0	c_1	c_2	\dots	c_k
$q_2 = r_1(1)/c_0$	0	c_0	c_1	\dots	c_{k-1}	c_k
	$r_1(1)$	\dots	$r_{k-1}(1)$	$r_k(1)$		
	c_0	\dots	c_{k-2}	c_{k-1}	c_k	
	\dots	$r_1(k-1)$	$r_2(k-1)$	\dots	$r_k(k-1)$	
$q_k = r_1(k-1)/c$	c_0	c_1	\dots	c_{k-1}	c_k	
	$r_1(k)$	\dots	$r_{k-1}(k)$	$r_k(k)$		

Table 1. Long division of polynomials.

EXAMPLE 3.

We will consider the same problem as in Example 1. The integral $I(1)$ will be determined using (3.4).

1) Use Table 1 to determine $Q(1,z)$ and $R(1,z)$

$$q = \begin{matrix} p_0 & p_1 \\ 0 & 0 \\ 0 & 0 \\ c_0 & c_1 \end{matrix}$$

$$q_1 = \begin{matrix} p_1 - p_0 c/c & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ c_0 & c_1 \end{matrix} \quad \begin{matrix} p_1 - p_0 c/c \\ 0 \\ 1 \\ 0 \\ 0 \\ c_0 \end{matrix} \quad \begin{matrix} c_1 \\ -(p_1 - p_0 c/c)c/c \\ 0 \\ 1 \\ 0 \end{matrix}$$

Hence $Q_1(1,z) = p_0/c_0 z + (p_1 - p_0 c/c)/c_0$

$$R_1(1,z) = -(p_1 - p_0 c/c)c/c_0$$

2) Use another table similar to Astrom(1970), called table 2 ,to determine c_i^j , r_i^j and p_i^j .

$$\begin{matrix} c_0 & c_1 & p_0 & p_1 & 0 \\ c_1 & c_0 & c_1 & c_0 & c_1 \\ c_0^{-1} & 1 & 1 & 0 & 0 \end{matrix} \left| \begin{matrix} p_1 & 0 \\ c_0 & c_1 \\ (p_1 - p_0 c/c)c/c_0 & c_0 \\ 0 & 1 \\ 0 & 1 \end{matrix} \right. \begin{matrix} -(p_1 - p_0 c/c)c/c_0 \\ c_0 \\ 1 \\ 0 \\ 0 \end{matrix}$$

The last row gives c_0^0 , p_0^0 and r_0^0 .

3) Equation (3.4) now gives

$$I_1(1) = q_1 \frac{p_0}{c_0} + \frac{1}{c_0} \left[\frac{p_1 r_0^0}{c_0} + \frac{p_0^0 r_0^0}{c_0} \right] = [p_0 p_1 (c_0^2 + c_1^2) - (p_0^2 + p_1^2)c_0 c_1] / [c_0^2 (c_0^2 + c_1^2)] \quad \square$$

Summary

Summing up, we get the following algorithm for solving the factorization problem (1.3).

- 1) Given $A(z)$, $B(z)$ and ρ .
- 2) Solve the Newton iteration equation (2.1) with the initial polynomial

$$C_0(z) = 1$$

It follows from (2.5) that

$$C X_1 = 2 \varrho a$$

$$C X_2 = 2 b$$

and

$$X = X_1 + X_2$$

It follows from (2.1) that it is sufficient to find x_i for $i=0$ to $k-2$ if the order of $A(z)$ is k . The variable x_k can be obtained from the $(k+1)$ th row. and x_{k-1} is then obtained by substitution.

3) X is given by $I(1)$, see (2.8) where $I(1)$ is given by recursive formulae (3.4). It is convenient to use Table 1 and Table 2 to solve this problem.

EXAMPLE 4.

Given $\varrho = 3$ and

$$A(z) = z^3 + 3z^2 + 3z + 3$$

$$B(z) = z^3 + 3z^2 + 3z + 3$$

Factorize $P(z)$ by means of the program called REFAC1 listed in appendix A.

step	c_0	c_1	c_2	c_3
0	1.000000	0.000000	0.000000	0.000000
1	56.500000	84.000010	48.000000	12.000000
2	29.191400	42.015720	24.027940	6.012446
3	16.140390	21.132080	12.227490	3.099148
4	10.049080	11.136740	7.002997	1.913081
5	7.291217	7.128360	5.556308	1.719164
6	6.305313	6.154831	5.727749	1.878278
7	6.041695	6.018396	5.958360	1.981686
8	6.001110	6.000490	5.998890	1.999510
9	6.000001	6.000000	6.000000	2.000000
10	6.000000	6.000000	6.000000	2.000000
11	6.000000	6.000000	6.000000	2.000000
12	6.000000	6.000000	6.000000	2.000000

The resultant polynomial $P(z)$ is given by

$$P(z) = 6z^3 + 6z^2 + 6z + 2$$

The number of steps required for convergence depends on the initial condition. Convergence is faster if $C_0(z)$ is closer to $P(z)$ as is seen in the following table.

step	c_0	c_1	c_2	c_3
0	10.583000	7.937253	4.535573	1.133893
1	7.416798	6.534993	4.937915	1.473130
2	6.382465	6.171319	5.626061	1.823389
3	6.063704	6.028771	5.936308	1.971217
4	6.002530	6.001125	5.997470	1.998875
5	6.000004	6.000002	5.999996	1.999998
6	6.000000	6.000000	6.000000	2.000000
7	6.000000	6.000000	6.000000	2.000000
8	6.000000	6.000000	6.000000	2.000000
9	6.000000	6.000000	6.000000	2.000000
10	6.000000	6.000000	6.000000	2.000000
11	6.000000	6.000000	6.000000	2.000000
12	6.000000	6.000000	6.000000	2.000000

This example is not a real system, but it shows that the method has a high precision. Another result obtained by a method of elimination is given for comparison.

step	c_0	c_1	c_2	c_3
0	10.583000	7.937253	4.535573	1.133893
1	7.416800	6.534993	4.937914	1.473130
2	6.382465	6.171319	5.626060	1.823389
3	6.063703	6.028772	5.936308	1.971218
4	6.002529	6.001125	5.997470	1.998876
5	6.000006	6.000002	5.999996	1.999997
6	6.000000	6.000000	6.000000	2.000000
7	6.000002	6.000000	6.000000	1.999999
8	6.000000	5.999999	6.000001	2.000000
9	6.000001	6.000001	5.999999	2.000000
10	6.000000	6.000000	6.000001	2.000000
11	6.000001	6.000000	6.000000	2.000000
12	6.000002	6.000000	5.999999	1.999999

4. REDUCED_VERSION

The disadvantage of the algorithm given in section 3 is that it requires computation time more than some other methods. This problem can be solved by a modified algorithm. Consider

$$\begin{aligned} \rho A(z)A(z^{-1}) + B(z)B(z^{-1}) &= s_0 + s_1(z+z^{-1}) + \dots + s_k(z+z^{-k}) \\ &= (s_k z^k + s_{k-1} z^{2k-1} + \dots + s_0 z^k + \dots + s_k) z^{-k} \\ &= S(z) z^{-k} \end{aligned} \quad (4.1)$$

The integral (1.7) becomes

$$\begin{aligned} I(\lambda) &= \frac{1}{2\pi i} \int \frac{\rho A(z)A(z^{-1}) + B(z)B(z^{-1})}{C(z)C(z^{-1})} \frac{dz}{z^{\lambda+1}} \\ &= \frac{1}{2\pi i} \int \frac{S(z)z^{-k}}{C(z)C(z^{-1})} \frac{dz}{z^{\lambda+1}} \\ &= \frac{1}{2\pi i} \int \left[\frac{S_q(z)z^{-k}}{C(z^{-1})} + \frac{S_r(z)z^{-k}}{C(z)C(z^{-1})} \right] \frac{dz}{z^{\lambda+1}} \\ &= \frac{1}{2\pi i} \int \frac{S_r(z)z^{-k}}{C(z)C(z^{-1})} \frac{dz}{z^{\lambda+1}} \end{aligned} \quad (4.2)$$

where

$$S_q(z)C(z) + S_r(z) = S(z) \quad (4.3)$$

$$S_q(z) = s_{q0}z^k + s_{q1}z^{k-1} + \dots + s_{qk-1}z$$

$$S_r(z) = s_{r0}z^k + s_{r1}z^{k-1} + \dots + s_{rk-1}z + s_{rk}$$

or in matrix form

$$\begin{bmatrix} c_0 & & & & & & & & \\ c_1 & c_0 & & & & & & & \\ & c_k & c_{k-1} & \dots & c_1 & & & & \\ 0 & c_k & \dots & c_2 & 0 & 1 & & & \\ & & \dots & & & & & & \\ & & & & & & 0 & 0 & 0 & \dots & 1 \\ & & & & & & & & & & & & s_{q0} & s_1 & \dots & s_k & s_{k-1} & \dots & s_0 \end{bmatrix} = \begin{bmatrix} s_{q0} \\ s_{q1} \\ \dots \\ s_{r0} \\ s_{r1} \\ \dots \\ s_{rk} \end{bmatrix}$$

The procedure now follows the previous one. Define

$$\underline{Q}(\lambda, z) C(z) + R(\lambda, z) = z^k \underline{S}(\lambda, z) \tag{4.4}$$

Replacing $R(\lambda, z)$ and $B(z)$ by $R(\lambda, z)$ and z^k in (3.2) gives

$$I_k(\lambda) = [\underline{Q}(\lambda, 0) + r_0^0 / c_0^0] / c_0^k \tag{4.5}$$

There is a general formula for r_0^k . Assume

$$R_k(\lambda, z) = r_1 z^{k-1} + r_2 z^{k-2} + \dots + r_k$$

It follows from the reduction rule that

$$R_{k-1}(\lambda, z) = z^{-1} [R_k(\lambda, z) - Y_k C^*(z)] \quad Y_k = r_k^k(\lambda) / c_0^k$$

Furthermore

$$r_0^k(\lambda) = k r_1(\lambda) + k r_2(\lambda) + \dots + k r_k(\lambda)$$

where

$$k_1 = -c_1^1 / c_0^1$$

$$k_2 = - [c_2^2 + c_1^2 k] / c_0^2$$

...

$$k_k = - [c_k^k + c_{k-1}^k k_1 + c_{k-2}^k k_2 + \dots + c_1^k k_{k-1}] / c_0^k$$

These equations can be written as

$$r_0^k(\lambda) = \sum_{i=1}^k k_i r_i(\lambda) \tag{4.6}$$

$$k = - \left(\sum_{j=1}^i c^i k_{j-i-j} \right) / c_0^i \quad \text{for } i=1 \text{ to } k \quad (4.7)$$

with $k_0 = 1$.

EXAMPLE 5.

Consider the same problem as in Example 1. We have

$$S(z) = p_0 p_1 z^2 + (p_0^2 + p_1^2)z + p_0 p_1 = s_0 z^2 + s_1 z + s_0$$

Determine $I(0)$ and $I(1)$ by use of the recursive formulas (4.5)

1) Use a table to determine $Q_s(0,z)$, $Q_s(1,z)$, $R_s(0,z)$ and $R_s(1,z)$.

$$q_0 = \begin{matrix} s_0 & s_1 & s_0 \\ 0 & c_1 & c_0 \end{matrix}$$

$$q_1 = \begin{matrix} s^{-1} q_0 c \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix} / c$$

$$q_2 = \begin{matrix} s^{-1} [s^{-1} q_0 c / c] c / c \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{matrix} / c$$

$$Q_s(0,z) = q_0 z + q_1$$

$$Q_s(1,z) = q_0 z^2 + q_1 z + q_2$$

$$R_s(0,z) = s_0 - q_1 c = r_1(0)$$

$$R_s(1,z) = -q_2 c = r_1(1)$$

2) Use table 2 to determine c_i^i

$$\begin{matrix} c_0^0 & c_1^1 \\ c_1^0 & c_0^0 \end{matrix}$$

$$c_0^{-1} c_1 / c_0$$

3) From (4.5), we obtain

$$\begin{aligned}
I(0) &= \{ q_1 + [r_1(0)c/c] / [c - c_1c/c] \} / c_0 \\
&= [(p_0^2 + p_1^2)c - 2p_0p_1c] / [c(c^2 - c_1^2)] \\
I(1) &= \{ q_2 + [r_1(1)c/c] / [c - c_1c/c] \} / c_0 \\
&= [p_0p_1(c_1^2 + c^2) - (p_0^2 + p_1^2)c_1c] / [c_0^2(c^2 - c_1^2)]
\end{aligned}$$

The Simplified Algorithm

A reduced algorithm is now obtained as follows

- 1) Multiply $S(z) = z^k [\rho A(z)A(z^{-1}) + B(z)B(z^{-1})]$
- 2) Obtain $S_r(z)$ (4.3), $Q_s(l, z)$ and $R_s(l, z)$ (4.4).
- 3) Calculate $r_0^0(l)$ and $I(l)$, (4.6) and (4.5).
- 4) Obtain $X_j(z)$ and $C_{j+1}(z)$, (2.8).

This algorithm takes $3.5n^2 + 1.5n - 3$ multiplication and divisions if x_{n-1} and x_n are determined by back substitution, where n is the order of the system.

EXAMPLE 6.

Given $\rho=1$ and

$$A(z) = z^3 - 1.60z^2 + 1.61z - 0.776$$

$$B(z) = z^2 - 0.95z + 0.2$$

Factorize $P(z)$ by means of the program called REFAC2 listed in appendix B.

Step	c_0	c_1	c_2	c_3
0	1.000	0.000	0.000	0.000
1	5.052	-6.732	3.082	-0.776
2	3.089	-3.860	1.736	-0.455
3	2.283	-2.633	1.234	-0.370
4	1.946	-2.187	1.158	-0.395
5	1.818	-2.073	1.194	-0.425
6	1.782	-2.058	1.219	-0.435
7	1.775	-2.061	1.226	-0.437
8	1.775	-2.062	1.228	-0.437
9	1.775	-2.063	1.228	-0.437
10	1.775	-2.063	1.228	-0.437
11	1.775	-2.063	1.228	-0.437
12	1.775	-2.063	1.228	-0.437

The resultant polynomial is given by

$$P(z) = 1.775z^3 - 2.063z^2 + 1.228z - 0.437$$

The accuracy of this algorithm is not as good as with the previous algorithm but it is acceptable in practice. The result obtained by the reduced version to the same numerical example in section 3 is listed as follows.

step	c_0	c_1	c_2	c_3
0	10.583000	7.937253	4.535573	1.133893
1	7.424470	6.536308	4.936104	1.472308
2	6.351079	6.175473	5.642726	1.829136
3	6.019774	6.039279	5.977029	1.984860
4	5.994448	5.998134	6.004982	2.001781
5	6.000621	5.999140	5.998795	1.999791
6	6.000024	6.000298	6.000184	1.999992
7	5.999967	5.999946	5.999983	2.000011
8	6.000010	6.000007	6.000001	1.999997
9	5.999997	6.000000	6.000001	2.000001
10	6.000000	5.999999	6.000001	2.000000
11	6.000000	6.000000	6.000000	2.000000
12	6.000000	5.999997	6.000000	2.000000

5. REFERENCES

1. Astrom K.J. (1970): Introduction to Stochastic Control Theory. Academic Press, New York.
2. Astrom K.J. (1979): Algebraic System Theory as a Tool for Regulator Design. Report CODEN:LUTFD2/(CTFRT-7164)/1-023/ (1979)
3. Astrom K.J., Jory E.I., and Agniel R.G. (1970): A Numerical Method for the Evaluation of Complex Integrals. IEEE Trans. AC-70, 468-471 Aug. 1970.
4. Kucera V. and Vostry Z. (1976): Expanding Spectral Density into Correlation Sequence. IEEE Trans. AC-76, 592-593 Aug. 1976.
5. Kucera V. (1979): Discrete Linear Control. The Polynomial Equation Approach. Academia Prague.
6. Murthy D.N.P. (1973): Factorization of Discrete-process Spectral Matrices. IEEE Trans. IT-73, 693-696 Sept. 1973.

Program factorization;

```
{ File is called REFAC1.
  Author Z.Y.Zhou.
  Date Apr.81.
```

The program uses Newton's iteration and recursive formulae of integral I (1) to perform the spectral factorization.

$$P(z)*P(z^{-1}) = r*A(z)*A(z^{-1}) + B(z)*B(z^{-1}).$$

The recursive relationship is given by

$$C_i(z)*X_i(z^{-1}) + X_{i-1}(z)*C_{i-1}(z^{-1}) = P(z)*P(z^{-1})$$

$$C_{i+1}(z) = (C_i(z) + X_i(z)) / 2$$

where i indicates the i-th iteration.

The solution of X(z) is obtained by

$$x_i = 2*c_i * I(i) + 2*c_{i-1} * I(i-1) + \dots + c_1 * I(0)$$

where I (1) is an integral of function $f(z)$ along the unit circle

$$f(z) = \frac{P(z)*P(z^{-1})}{(C(z)*C(z^{-1}))}$$

```
const n=10;
      m=50;
      l=8;
      eps=0.001;
```

```
var a,b,t,bt,ct,c,r,x:array[0..n] of real;
    c:array[0..m,0..n] of real;
    qa,qb,int,fra,frb:array[0..l] of real;
    ra,rb:array[0..l,0..n] of real;
    na,ni,la,i,j,s,p:integer;
    stab:array[0..m] of integer;
    fa,fb,fc,r:real;
    ch:char;
    filename:array[1..n] of char;
```

```
{-----Input}
Procedure Input;
{ Input Na -- degree of A(z);
  A[i] -- element of A(z);
  Ni -- maximum number of iteration;
  filename -- the name of a file for store results. }
begin
  writeln('Input Ni (the number of iteration) and filename.');
```

```
  read(ni,filename);
  writeln('Input Na (the degree of A(z))');
```

```
  read(na);
  repeat
    writeln('Input A[i]');
```

```
    for i:=0 to na do read(a[i]);
    writeln('Input B[i]');
```

```

for i:=0 to na do read(b[i]);
writeln('Input r (the weighting coefficient)');
read(r);
writeln('Please check A[i],rewrite?');
repeat read(ch) until ch<>' ';
until ch<>'Y';
end; { of input }

{-----Initialize}
Procedure Initialize;
begin
  writeln(' Set initial C[z] ?');
  repeat read(ch) until ch<>' ';
  if ch='Y', then
    for i:=0 to na do read(c[i])
  else begin
    for i:=1 to na do c[i]:=0;
    c[0,0]:=1;
  end;
  la:=na-2;
  for i:=0 to ni do stab[i]:=0;
end; { of initialize }

{-----Polydivision}
Procedure Polydivision(k:integer);
begin
  qa[0]:=a[0]/c[k,0];
  qb[0]:=b[0]/c[k,0];
  for i:=1 to na do begin
    ra[0,i]:=a[i]-qa[0]*c[k,i];
    rb[0,i]:=b[i]-qb[0]*c[k,i];
  end;
  int[0]:=0;
  if la>0 then
    for s:=1 to la do begin
      qa[s]:=ra[s-1,1]/c[k,0];
      qb[s]:=rb[s-1,1]/c[k,0];
      ra[s-1,na+1]:=0;
      rb[s-1,na+1]:=0;
      for i:=1 to na do begin
        ra[s,i]:=ra[s-1,i+1]-qa[s]*c[k,i];
        rb[s,i]:=rb[s-1,i+1]-qb[s]*c[k,i];
      end;
      ra[s,0]:=0;
      rb[s,0]:=0;
      int[s]:=r*qa[s]*a[0]+qb[s]*b[0];
    end;
  end; { of polydivision }

{-----Iteration}
Procedure Iteration(k:integer);
var crm,int1:real;
    laa:integer;
begin
  for i:=0 to na do begin
    ct[i]:=c[k,i];
    at[i]:=a[i];
    bt[i]:=b[i];
  end;
  for j:=0 to na-1 do begin
    int1:=r*at[na-j]*at[na-j]+bt[na-j]*bt[na-j];

```

```

int[0]:=int[0]+int1/ct[0];
if la>0 then for s:=1 to la do begin
  int1:=r*at[na-j]*ra[s,na-j]+bt[na-j]*rb[s,na-j];
  int[s]:=int[s]+int1/ct[0];
end;
for i:=0 to na-j do cr[i]:=ct[na-j-i];
if cr[na-j]<>0 then crm:=cr[na-j] else crm:=epsi;
fc:=ct[na-j]/crm;
fa:=at[na-j]/crm;
fb:=bt[na-j]/crm;
if la>0 then for s:=1 to la do begin
  fra[s]:=ra[s,na-j]/crm;
  frb[s]:=rb[s,na-j]/crm;
end;
for i:=0 to na-j-1 do begin
  ct[i]:=ct[i]-fc*cr[i];
  at[i]:=at[i]-fa*cr[i];
  bt[i]:=bt[i]-fb*cr[i];
  if la>0 then for s:=1 to la do begin
    ra[s,i]:=ra[s,i]-fra[s]*cr[i];
    rb[s,i]:=rb[s,i]-frb[s]*cr[i];
  end;
end;
end;

if ct[0]<0 then stab[k]:=stab[k]-1;
if j=na-1 then begin
  int1:=r*at[na-j-1]*at[na-j-1]+bt[na-j-1]*bt[na-j-1];
  int[0]:=int[0]+int1/ct[0];
  if la>0 then for s:=1 to la do begin
    int1:=r*at[na-j-1]*ra[s,na-j-1]+bt[na-j-1]*rb[s,na-j-1];
    int[s]:=int[s]+int1/ct[0];
  end;
end;
end;

if la<0 then laa:=0 else laa:=la;
for s:=0 to laa do begin
  int[s]:=int[s]/c[k,0];
  x[s]:=c[k,s]*int[0];
  if (la>0) and (s>0) then
    for i:=1 to s do x[s]:=x[s]+2*c[k,s-i]*int[i];
end;
x[na]:=2*(r*a[0]*a[na]+b[0]*b[na])-c[k,na]*x[0]/c[k,0];
if na>1 then begin
  x[na-1]:=2*(r*(a[0]*a[na-1]+a[1]*a[na])+b[0]*b[na-1]+b[1]*b[na]);
  if na>2 then
    x[na-1]:=x[na-1]-c[k,na-1]*x[0]-c[k,na]*x[1]-c[k,1]*x[na]/c[k,0]
  else x[1]:=x[na-1]-c[k,1]*x[0]-c[k,1]*x[na]/(c[k,0]+c[k,2]);
end;
end; { of iteration }

{-----Store}
Procedure Store;
var len:integer;
    outfile:file of char;
begin
  len:=--1;
  rewrite(outfile,filename,'dat',len);
  if len=-1 then writeln('illfile');
  writeln(outfile,'Spectral Factorization:');
  write(outfile,
' ');

```

```

for i:=0 to na do write(outfile,' [',i:2,'] ');
writeln(outfile); writeln(outfile);
write(outfile,' A[z] ');
for i:=0 to na do write(outfile,a[i]:8:4);
writeln(outfile);
write(outfile,' B[z] ');
for i:=0 to na do write(outfile,b[i]:8:4);
writeln(outfile); writeln(outfile);
writeln(outfile,' r= ',r:8:4);
writeln(outfile);
for j:=0 to ni do begin
  write(outfile,j:6);
  for i:=0 to na do write(outfile,c[j,i]:8:4);
  if stab[j]<0 then write(' unstable ');
  writeln(outfile);
end;
close(outfile);
end; { of store }

{=====Code of main program}
begin
  input;
  initialize;
  for p:=0 to ni do begin
    polydivision(p);
    iteration(p);
    for i:=0 to na do c[p+1,i]:=((c[p,i]+x[i])/2);
  end;
  store;
end. { of main program }

```

Program factorization!

{ File is called REFACT2.
 Author Z.Y.Zhou.
 Date Apr.81.

The program uses Newton's iteration and recursive fomulae of integral I (1) to perform the spectral factorization.

$$P(z)*P(z) = r*A(z)*A(z) + B(z)*B(z).$$

$$S(z)*z^{-n}$$

The recursive relationship is given by

$$C_i(z)*X_i(z) + X_{i-1}(z)*C_i(z) = P_i(z)*P_i(z)$$

$$C_{i+1}(z) = (C_i(z) + X_i(z)) / z$$

where i indicates the i-th iteration.

$$P(z) = \lim_i C_i(z)$$

The solution of X(z) is obtained by

$$x_i = 2*c_i * I(i) + 2*c_{i-1} * I(i-1) + \dots + c_i * I(0)$$

where I (1) is an integral of function f (z) along the unit circle

$$f(z) = S_k(z)*z^{-k} / (C_k(z)*C_k(z))$$

```
const n=10;
      m=50;
      eps=0.001;
var a,b,s,r,x,q;int:array[0..n] of real;
      c:array[0..m,0..n] of real;
      rrs:array[0..n,0..n] of real;
      na,nj,i,j,l,p:integer;
      stab:array[0..m] of integer;
      r:real;
      ch:char;
      filename:array[1..n] of char;
```

```
{-----Input}
Procedure Input;
{ Input Na -- degree of A(z);
  A[i] -- element of A(z);
  Ni -- maximum munber of iteration;
  filename -- the name of a file for store results. }
begin
  writeln('Input Ni (the number of iteration) and filename.');
```

```
  read(ni,filename);
  writeln('Input Na (the degree of A(z))');
```

```
  read(na);
  repeat
    writeln('Input A[i]');
```

```

for i:=0 to na do read(a[i]);
writeln('Input B[i]');
for i:=0 to na do read(b[i]);
writeln('Input r (the weighting coefficient)');
read(r);
writeln('Please check A[i],rewrite?');
repeat read(ch) until ch<>' ';
until ch<>'y';
end; { of input }

```

```

{-----Initialize}
Procedure Initialize;
begin
  writeln(' Set initial C[z] ?');
  repeat read(ch) until ch<>' ';
  if ch='y' then
    for i:=0 to na do read(c[0,i])
    else begin
      for i:=1 to na do c[0,i]:=0;
      c[0,0]:=1;
    end;
  for i:=0 to ni do stab[i]:=0;
end; { of initialize }

```

```

{-----polymultiply}
Procedure polymultiply;
var aa,bb:array[0..n] of real;
begin
  for i:=0 to na do begin
    aa[i]:=0;
    bb[i]:=0;
    for j:=i to na do begin
      aa[i]:=aa[i]+a[j-i]*a[j];
      bb[i]:=bb[i]+b[j-i]*b[j];
    end;
    s[na-i]:=r*aa[i]+bb[i];
  end;
end; { of polymultiply }

```

```

{-----polyfactor}
Procedure Polyfactor(k:integer);
var sq:array[0..n] of real;
begin
  sq[0]:=s[0]/c[k,0];
  for i:=1 to na-1 do begin
    sq[i]:=sq[i];
    for j:=1 to i do
      sq[i]:=sq[i]-sq[i-j]*c[k,j];
    sq[i]:=sq[i]/c[k,0];
    write(sq[i]:8:4);
  end;
  for i:=0 to na-1 do begin
    sr[i]:=s[na-i];
    for j:=0 to na-i-1 do
      sr[i]:=sr[i]-sq[j+i]*c[k,na-j];
    end;
    sr[na]:=s[0];
  end; { of polyfactor }
{-----Polydivision}
Procedure Polydivision(k:integer);

```

```

begin
  q[0]:=sr[0]/c[k,0];
  for i:=1 to na do
    rrs[0,i]:=sr[i]-q[0]*c[k,i];
  int[0]:=q[0];
  for l:=1 to na do begin
    q[l]:=rrs[l-1,1]/c[k,0];
    rrs[l-1,na+1]:=0;
    for i:=1 to na do
      rrs[l,i]:=rrs[l-1,i+1]-q[l]*c[k,i];
    int[l]:=q[l];
  end;
end; of polydivision }

{-----Iteration}
Procedure Iteration(k:integer);
var cr,f,array[0..n] of real;
ct:array[0..n,0..n] of real;
int1:real;
begin
  for i:=0 to na do
    ct[0,i]:=c[k,i];
  for j:=0 to na-1 do begin
    for i:=0 to na-j do cr[i]:=ct[j,na-j-i];
    f[j]:=ct[j,na-j]/cr[na-j];
    for i:=0 to na-j-1 do
      ct[j+1,i]:=ct[j,i]-f[j]*cr[i];
    if ct[j,0]<0 then stab[k]:=stab[k]-1;
  end;
  af[l]:=-f[na-1];
  for i:=2 to na do
    for j:=1 to i-1 do
      af[i]:=-f[na-i]-af[i-j]*ct[na-i,j]/ct[na-i,0];
  for l:=0 to na do begin
    int1:=0;
    for j:=1 to na do
      int1:=int1+af[j]*rrs[l,j];
    int1:=int1/ct[na,0];
    int[l]:=int[l]+int1;
  end;
  for l:=0 to na do begin
    int[l]:=int[l]/c[k,0];
    x[l]:=c[k,l]*int[0];
    for i:=1 to l do
      x[l]:=x[l]+2*c[k,l-i]*int[i];
  end;
end; { of iteration }

{-----Store}
Procedure Store;
var len:integer;
    outfile:file of char;
begin
  len:=1;
  rewrite(outfile,filename,'dat',len);
  if len=-1 then writeln('illfile');
  writeln(outfile,'Spectral Factorization:');
  write(outfile,
    ' ');
  for i:=0 to na do write(outfile,'  ',i:2,' ');
  writeln(outfile); writeln(outfile);

```

```

write(outfile,' A[z] ');
for i:=0 to na do write(outfile,a[i]:8:4);
writeln(outfile);
write(outfile,' B[z] ');
for i:=0 to na do write(outfile,b[i]:8:4);
writeln(outfile); writeln(outfile);
writeln(outfile,' r= ',r:8:4);
writeln(outfile);
for j:=0 to ni do begin
  write(outfile,j:6);
  for i:=0 to na do write(outfile,c[j,i]:8:4);
  if stab[j]<0 then write(' unstable ');
  writeln(outfile);
end;
close(outfile);
end; { of store }

{=====Code of main program}
begin
input;
polymultiply;
initialize;
for p:=0 to ni do begin
  polyfactor(p);
  polydivision(p);
  iteration(p);
  for i:=0 to na do c[p+1,i]:=c[p,i]+x[i]/2;
end;
store;
end. { of main program }

```