The Schur Algorithm

The Schur algorithm is related to the Levinson algorithm. It is also a fast recursion to solve the Yule-Walker equations, with about the same computational complexity as the Levinson algorithm (order p^2). However, it is numerically more stable, and can be run more efficiently on parallel hardware.

See also:

- B. Porat, *A course in digital signal processing*, Wiley: chapter 13.4
- M.H. Hayes, *Statistical digital signal processing and modeling*, Wiley, 1996: chapter 5.2.6

Recapitulation

The Yule-Walker equations

Given a correlation sequence $\{r_x(0), \cdots, r_x(p)\}$, we need to solve for the filter coefficients $a_p(1), \cdots, a_p(p)$ and innovation noise power ϵ_p in

$$\begin{bmatrix} r_x(0) & r_x(1) & r_x(2) & \cdots & r_x(p) \\ r_x(1) & r_x(0) & r_x(1) & \cdots & r_x(p-1) \\ r_x(2) & r_x(1) & r_x(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & r_x(1) \\ r_x(p) & r_x(p-1) & \cdots & r_x(1) & r_x(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_p(1) \\ a_p(2) \\ \vdots \\ a_p(p) \end{bmatrix} = \begin{bmatrix} \epsilon_p \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The matrix $\mathbf{R}_x$ is assumed to be (strictly) positive definite.

Recapitulation—The Levinson algorithm

The problem can be extended as follows:

$$\begin{bmatrix} r_x(0) & r_x(1) & r_x(2) & \cdots & r_x(p) \\ r_x(1) & r_x(0) & r_x(1) & \cdots & r_x(p-1) \\ r_x(2) & r_x(1) & r_x(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & r_x(1) \\ r_x(p) & r_x(p-1) & \cdots & r_x(1) & r_x(0) \end{bmatrix} \begin{bmatrix} 1 & a_p(p) \\ a_p(1) & a_p(p-1) \\ \vdots & \vdots \\ a_p(p-1) & a_p(1) \\ a_p(p) & 1 \end{bmatrix} = \begin{bmatrix} \epsilon_p & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & \epsilon_p \end{bmatrix}$$

Given the solution for p, we look for a solution for $p+1$. We first try:

$$\begin{bmatrix} r_x(0) & r_x(1) & r_x(2) & \cdots & r_x(p+1) \\ r_x(1) & r_x(0) & r_x(1) & \cdots & r_x(p) \\ r_x(2) & r_x(1) & r_x(0) & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & r_x(1) \\ r_x(p+1) & r_x(p) & \ddots & r_x(1) & r_x(0) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ a_p(1) & a_p(p) \\ \vdots & \vdots \\ a_p(p) & a_p(1) \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \epsilon_p & \gamma_p \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \gamma_p & \epsilon_p \end{bmatrix}$$

Calculation of γ_p requires an inner product:

$\gamma_p = [r_x(p+1),\ r_x(p),\ \cdots,\ r_x(1)][1,\ a_p(1),\ a_p(p),\ \cdots,\ a_p(p)]^T.$

The Levinson algorithm

Step 2 is to rotate the pair of vectors at the LHS and RHS such that γ_p is cancelled:

$$\begin{bmatrix} r_x(0) & r_x(1) & r_x(2) & \cdots & r_x(p+1) \\ r_x(1) & r_x(0) & r_x(1) & \cdots & r_x(p) \\ r_x(2) & r_x(1) & r_x(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & r_x(1) \\ r_x(p+1) & r_x(p) & \cdots & r_x(1) & r_x(0) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ a_p(1) & a_p(p) \\ \vdots & \vdots \\ a_p(p) & a_p(1) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\rho_{p+1} \\ -\rho_{p+1} & 1 \end{bmatrix} = \begin{bmatrix} \epsilon_p & \gamma_p \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \gamma_p & \epsilon_p \end{bmatrix} \begin{bmatrix} 1 & -\rho_{p+1} \\ -\rho_{p+1} & 1 \end{bmatrix}$$

For $\rho_{p+1} = \frac{\gamma_p}{\epsilon_p}$, this will bring the RHS into the required form. The LHS is then the solution of the YW equations of order $p+1$. The updated solutions are:

$$\begin{bmatrix} 1 & 0 \\ a_p(1) & a_p(p) \\ \vdots & \vdots \\ a_p(p) & a_p(1) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\rho_{p+1} \\ -\rho_{p+1} & 1 \end{bmatrix} =: \begin{bmatrix} 1 & a_{p+1}(p+1) \\ a_{p+1}(1) & a_{p+1}(p) \\ \vdots & \vdots \\ a_{p+1}(p) & a_{p+1}(1) \\ a_{p+1}(p+1) & 1 \end{bmatrix} ; \quad \begin{bmatrix} \epsilon_p & \gamma_p \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \gamma_p & \epsilon_p \end{bmatrix} \begin{bmatrix} 1 & -\rho_{p+1} \\ -\rho_{p+1} & 1 \end{bmatrix} =: \begin{bmatrix} \epsilon_{p+1} & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & \epsilon_{p+1} \end{bmatrix}$$

The Schur algorithm

- The Levinson algorithm involves two times $2p$ multiplications and p additions. With p computational processors, this work can be done in parallel. However, the computation of γ_p (p additions) is not easily parallelized.
- The Schur algorithm is an alternative to Levinson to solve the same equations. It is based on the idea that we do not need the filter coefficients $\{a_p(1), \cdots a_p(p)\}$ for all p, the reflection coefficients $\{\rho_1, \cdots, \rho_p\}$ completely specify the filter.

The Schur algorithm

Consider the same YW equations, but now operating on an *infinite* matrix:

$$
\left[\begin{array}{cccc|ccc}
r_x(0) & r_x(1) & \ddots & r_x(p) & r_x(p+1) & \ddots & \ddots \\
r_x(1) & r_x(0) & r_x(1) & \ddots & r_x(p) & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
r_x(p) & \ddots & r_x(1) & r_x(0) & r_x(1) & \ddots & \ddots \\
\hline
r_x(p+1) & r_x(p) & \ddots & r_x(1) & r_x(0) & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{array}\right]
\left[\begin{array}{cc}
1 & a_p(p) \\
a_p(1) & \vdots \\
\vdots & a_p(1) \\
a_p(p) & 1 \\
\hline
0 & 0 \\
0 & 0 \\
\vdots & \vdots
\end{array}\right]
=
\left[\begin{array}{cc}
\epsilon_p & 0 \\
0 & \vdots \\
\vdots & 0 \\
0 & \epsilon_p \\
\hline
\gamma_p & * \\
* & * \\
\vdots & \vdots
\end{array}\right]
$$

Here, $*$ denotes an unknown number (typically nonzero).

The Schur algorithm

Step 1: shift

The right solution vector on the LHS is shifted down one position:

$$\left[\begin{array}{ccccc|cc} r_x(0) & r_x(1) & \ddots & r_x(p) & r_x(p+1) & \ddots & \ddots \\ r_x(1) & r_x(0) & r_x(1) & \ddots & r_x(p) & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ r_x(p) & \ddots & r_x(1) & r_x(0) & r_x(1) & \ddots & \ddots \\ r_x(p+1) & r_x(p) & \ddots & r_x(1) & r_x(0) & \ddots & \ddots \\ \hline \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{array}\right] \left[\begin{array}{cc} 1 & 0 \\ a_p(1) & a_p(p) \\ \vdots & \vdots \\ a_p(p) & a_p(1) \\ 0 & 1 \\ \hline 0 & 0 \\ \vdots & \vdots \end{array}\right] = \left[\begin{array}{cc} \epsilon_p & \gamma_p \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \gamma_p & \epsilon_p \\ \hline * & * \\ \vdots & \vdots \end{array}\right]$$

The corresponding vector on the RHS is also shifted by one position, but with a fill-in at the top (equal to γ_p)

The Schur algorithm

Step 2: rotate

$$\left[\begin{array}{ccccc|cc} r_x(0) & r_x(1) & \ddots & r_x(p) & r_x(p+1) & \ddots & \ddots \\ r_x(1) & r_x(0) & r_x(1) & \ddots & r_x(p) & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ r_x(p) & \ddots & r_x(1) & r_x(0) & r_x(1) & \ddots & \ddots \\ r_x(p+1) & r_x(p) & \ddots & r_x(1) & r_x(0) & \ddots & \ddots \\ \hline \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{array}\right] \left[\begin{array}{cc} 1 & 0 \\ a_p(1) & a_p(p) \\ \vdots & \vdots \\ a_p(p) & a_p(1) \\ 0 & 1 \\ \hline 0 & 0 \\ \vdots & \vdots \end{array}\right] \begin{bmatrix} 1 & -\rho_{p+1} \\ -\rho_{p+1} & 1 \end{bmatrix} = \left[\begin{array}{cc} \epsilon_p & \gamma_p \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \gamma_p & \epsilon_p \\ \hline * & * \\ \vdots & \vdots \end{array}\right] \begin{bmatrix} 1 & -\rho_{p+1} \\ -\rho_{p+1} & 1 \end{bmatrix}$$

$$=: \left[\begin{array}{cc} \epsilon_{p+1} & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & \epsilon_{p+1} \\ \hline * & * \\ \vdots & \vdots \end{array}\right]$$

As before, we find $\rho_{p+1} = \frac{\gamma_p}{\epsilon_p}$.

The Schur algorithm

The main observation is that *we do not need to keep track of* $\{a_p(i)\}$ *to compute the reflection coefficients.* It is sufficient to keep track of the evolution of the RHS:

$$\left[\begin{array}{cc} \epsilon_p & 0 \\ 0 & \vdots \\ \vdots & 0 \\ 0 & \epsilon_p \\ \hline \gamma_p & * \\ * & * \\ \vdots & \vdots \end{array}\right] \overset{\Longrightarrow}{\text{shift}} \left[\begin{array}{cc} \epsilon_p & \gamma_p \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \gamma_p & \epsilon_p \\ \hline * & * \\ \vdots & \vdots \end{array}\right] \overset{\Longrightarrow}{\text{rotate, } \rho_{p+1} = \frac{\gamma_p}{\epsilon_p}} \left[\begin{array}{cc} \epsilon_{p+1} & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & \epsilon_{p+1} \\ \hline * & * \\ \vdots & \vdots \end{array}\right] \overset{\Longrightarrow}{\text{shift}} \quad \cdots$$

To initialize the recursion ($p = 0$), note that

$$\left[\begin{array}{c|ccc} r_x(0) & r_x(1) & \ddots & \ddots \\ \hline r_x(1) & r_x(0) & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots \end{array}\right] \left[\begin{array}{cc} 1 & 1 \\ \hline 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \end{array}\right] = \left[\begin{array}{cc} r_x(0) & r_x(0) \\ \hline r_x(1) & r_x(1) \\ * & * \\ \vdots & \vdots \end{array}\right]$$

The Schur algorithm

We define the notation

$$\left[\begin{array}{cc} r_x(0) & r_x(0) \\ \hline r_x(1) & r_x(1) \\ * & * \\ \vdots & \vdots \end{array}\right] =: \left[\begin{array}{cc} g_0(0) & h_0(0) \\ \hline g_0(1) & h_0(1) \\ g_0(2) & h_0(2) \\ \vdots & \vdots \end{array}\right]$$

The Schur recursion then continues as

$$\left[\begin{array}{cc} g_0(0) & h_0(0) \\ \hline g_0(1) & h_0(1) \\ g_0(2) & h_0(2) \\ \vdots & \vdots \end{array}\right] \underset{\text{shift}}{\Longrightarrow} \left[\begin{array}{cc} g_0(0) & * \\ g_0(1) & h_0(0) \\ \hline g_0(2) & h_0(1) \\ \vdots & \vdots \end{array}\right] \underset{\text{rotate}}{\Longrightarrow} \left[\begin{array}{cc} g_1(0) & 0 \\ 0 & h_1(1) \\ \hline g_1(2) & h_1(2) \\ \vdots & \vdots \end{array}\right] \underset{\text{shift}}{\Longrightarrow} \left[\begin{array}{cc} g_1(0) & * \\ 0 & 0 \\ g_1(2) & h_1(1) \\ \hline \vdots & \vdots \end{array}\right] \underset{\text{rotate}}{\Longrightarrow} \left[\begin{array}{cc} g_2(0) & 0 \\ 0 & 0 \\ 0 & h_2(2) \\ \hline \vdots & \vdots \end{array}\right] \underset{\text{shi}}{\Longrightarrow}$$

Note that we don't need to keep track of the first row: The fill-in '$*$' is automatically cancelled after each rotation. Ignoring the first row is equivalent to setting $g_0(0) = 0$ in the beginning. The reflection coefficient $\rho_{p+1} = g_p(p+1)/h_p(p)$.

Polynomials

We can associate polynomials with these vectors:

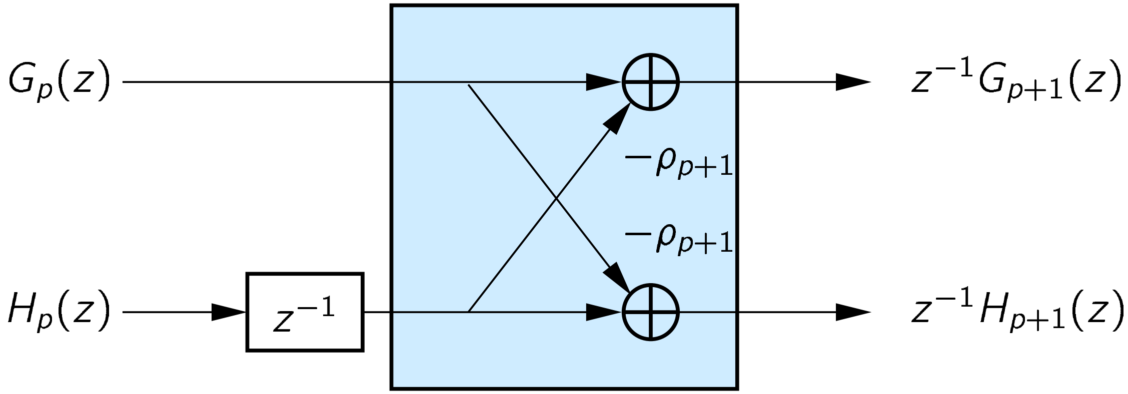
$$\begin{cases} G_0(z) = \qquad g_0(1)z^{-1} + g_0(2)z^{-2} + \cdots \\ H_0(z) = h_0(0) + h_0(1)z^{-1} + h_0(2)z^{-2} + \cdots \end{cases} \quad \text{and} \quad \begin{cases} G_p(z) = \qquad g_p(p+1)z^{-1} + g_p(p+2)z^{-2} + \cdots \\ H_p(z) = h_p(p) + h_p(p+1)z^{-1} + h_p(p+2)z^{-2} + \cdots \end{cases}$$

(Note that $g_0(0)$ is omitted, equivalent to setting $g_0(0) = 0$.)

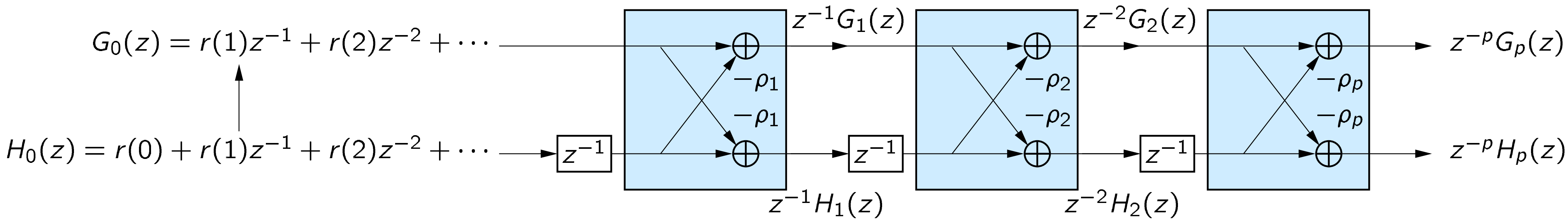
The same recursion, written in terms of the polynomials, is then

$$z^{-1}[G_{p+1}(z), \quad H_{p+1}(z)] \quad = \quad [G_p(z), \quad H_p(z)] \begin{bmatrix} 1 & 0 \\ 0 & z^{-1} \end{bmatrix} \begin{bmatrix} 1 & -\rho_{p+1} \\ -\rho_{p+1} & 1 \end{bmatrix}$$

where $\rho_{p+1} = \frac{g_p(p+1)}{h_p(p)}$.



The Schur algorithm



The Schur algorithm

Cholesky factorization of a Toeplitz matrix

For a positive matrix $\mathbf{C}$, the Cholesky factorization is a factorization as

$$\mathbf{C} = \mathbf{L}\mathbf{D}\mathbf{L}^T, \qquad \mathbf{L}\text{: lower triangular, } \mathbf{D}\text{: diagonal}$$

The Schur algorithm provides a factorization of the Toeplitz matrix $\mathbf{R}_x$, as follows.

Define the matrix $\mathbf{A}_p$ in terms of the solutions of the Yule-Walker equations of order 0 until p:

$$\mathbf{A}_p = \left[\begin{array}{c|c|c|c|c} 1 & a_1(1) & a_2(2) & \vdots & a_p(p) \\ & 1 & a_2(1) & \vdots & \vdots \\ & & 1 & \vdots & a_p(2) \\ & & & 1 & a_p(1) \\ & & & & 1 \end{array}\right]$$

Cholesky factorization

Recall that for any order p the Yule-Walker equation on the reverse sequence is

$$\begin{bmatrix} r_x(0) & r_x(1) & r_x(2) & \cdots & r_x(p) \\ r_x(1) & r_x(0) & r_x(1) & \cdots & r_x(p-1) \\ r_x(2) & r_x(1) & r_x(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & r_x(1) \\ r_x(p) & r_x(p-1) & \cdots & r_x(1) & r_x(0) \end{bmatrix} \begin{bmatrix} a_p(p) \\ \vdots \\ a_p(2) \\ a_p(1) \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \epsilon_p \end{bmatrix}$$

It follows that

$$\begin{bmatrix} r_x(0) & r_x(1) & r_x(2) & \cdots & r_x(p) \\ r_x(1) & r_x(0) & r_x(1) & \cdots & r_x(p-1) \\ r_x(2) & r_x(1) & r_x(0) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & r_x(1) \\ r_x(p) & r_x(p-1) & \cdots & r_x(1) & r_x(0) \end{bmatrix} \left[\begin{array}{c|c|c|c|c} 1 & a_1(1) & a_2(2) & \vdots & a_p(p) \\ & 1 & a_2(1) & \vdots & \vdots \\ & & 1 & \vdots & a_p(2) \\ & & & 1 & a_p(1) \\ & & & & 1 \end{array}\right] = \left[\begin{array}{c|c|c|c|c} \epsilon_0 & & & & \\ * & \epsilon_1 & & & \\ * & * & \epsilon_2 & & \\ * & * & * & * & \\ \vdots & \vdots & \vdots & \vdots & \epsilon_p \end{array}\right]$$

$$\Leftrightarrow \quad \mathbf{R}_x \mathbf{A}_p = \mathbf{E}$$

Cholesky factorization

Consider now $\mathbf{A}_p^T \mathbf{R}_x \mathbf{A}_p$. Because $\mathbf{A}_p^T$ and $\mathbf{R}_x \mathbf{A}_p$ are lower triangular matrices, it must be lower. But it is also a symmetric matrix. Hence it is a diagonal matrix. The entries on the main diagonal are seen to be $\{\epsilon_0, \cdots, \epsilon_p\}$.

We thus found

$$\mathbf{A}_p^T \mathbf{R}_x \mathbf{A}_p = \mathbf{D}_p \quad \text{(diagonal)} \qquad \Rightarrow \qquad \mathbf{R}_x = \mathbf{A}_p^{-T} \mathbf{D}_p \mathbf{A}_p^{-1}, \qquad \mathbf{R}_x^{-1} = \mathbf{A}_p \mathbf{D}_p^{-1} \mathbf{A}_p^T$$

These are Cholesky factorizations of $\mathbf{R}_x$ and $\mathbf{R}_x^{-1}$.

They are used in many applications.

The Schur-Cohn stability test

- A rational filter $H(z) = \frac{B(z)}{A(z)}$ is stable if all its poles are within the unit circle. This is determined by the zeros of $A(z)$.

- A filter $G(z)$ is an *allpass* filter if $|G(e^{j\omega}| = 1$.

  Property: a rational filter $G(z)$ is allpass if it has the form

$$G(z) = \frac{A^R(z)}{A(z)} = \frac{a(p) + a(p-1)z^{-1} + \cdots + z^{-p}}{1 + a(1)z^{-1} + \cdots + a(p)z^{-p}}$$

To test the stability of a filter $H(z)$, it is sufficient to form $G(z)$ and test its stability. We will do this recursively.

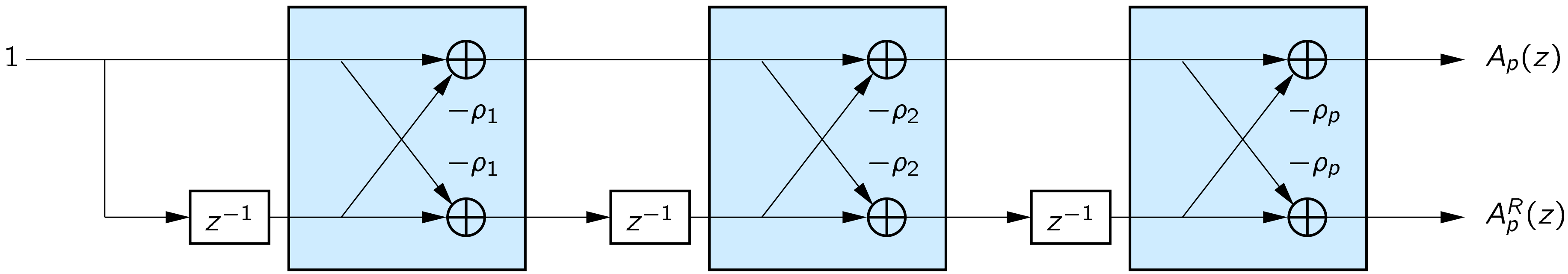
Let the poles of $G(z)$ be given by $\{p_1, \cdots, p_p\}$.

$$A(z) = (1 - p_1 z^{-1}) \cdots (1 - p_p z^{-1}) = 1 + \cdots + (p_1 p_2 \cdots p_p) z^{-p}$$

A necessary condition for stability is $|a(p)| = |p_1 p_2 \cdots p_p| < 1$.

The Schur-Cohn stability test

We will now try to interprete $A(z)$ and $A^R(z)$ as the transfer functions of a lattice filter as in the Levinson recursion:



We will need to compute the reflection coefficients, and also show that in fact it is a valid structure for these polynomials.

We start from $A_p(z)$ and $A_p^R(z)$ and work backwards.

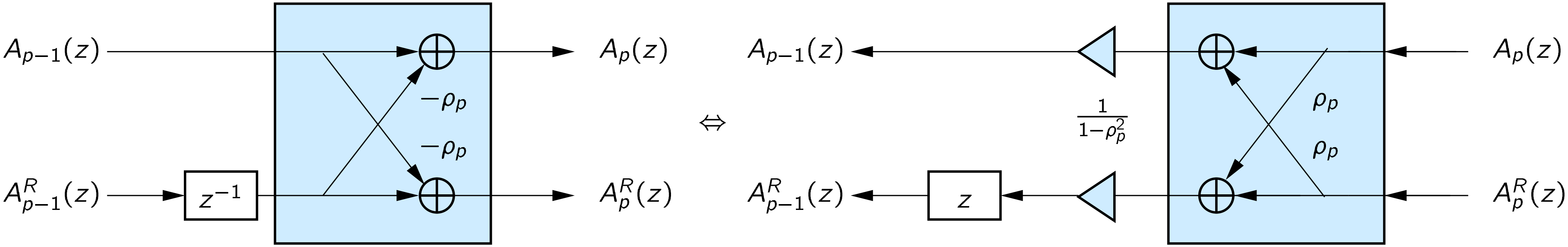
The Schur-Cohn stability test

- First step: given $[A_p(z),\ A_p^R(z)]$, compute $[A_{p-1}(z),\ A_{p-1}^R(z)]$. We have, in the filter structure:

$$\begin{bmatrix} A_p(z) \\ A_p^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -\rho_p \\ -\rho_p & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & z^{-1} \end{bmatrix} \begin{bmatrix} A_{p-1}(z) \\ A_{p-1}^R(z) \end{bmatrix}$$

Hence

$$\begin{bmatrix} A_{p-1}(z) \\ A_{p-1}^R(z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & z \end{bmatrix} \begin{bmatrix} 1 & -\rho_p \\ -\rho_p & 1 \end{bmatrix}^{-1} \begin{bmatrix} A_p(z) \\ A_p^R(z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & z \end{bmatrix} \frac{1}{1-\rho_p^2} \begin{bmatrix} 1 & \rho_p \\ \rho_p & 1 \end{bmatrix} \begin{bmatrix} A_p(z) \\ A_p^R(z) \end{bmatrix}$$



The Schur-Cohn stability test

- In terms of sequences, the first step is

$$\frac{1}{1-\rho_p^2}\begin{bmatrix} 1 & \rho_p \\ \rho_p & 1 \end{bmatrix}\begin{bmatrix} 1 & a_p(1) & \cdots & a_p(p-1) & a_p(p) \\ a_p(p) & a_p(p-1) & \cdots & a_p(1) & 1 \end{bmatrix}$$

- We choose the reflection coefficient such that a zero is created: with $\rho_p = -a_p(p)$:

$$\frac{1}{1-\rho_p^2}\begin{bmatrix} 1 & \rho_p \\ \rho_p & 1 \end{bmatrix}\begin{bmatrix} 1 & a_p(1) & \cdots & a_p(p-1) & a_p(p) \\ a_p(p) & a_p(p-1) & \cdots & a_p(1) & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & a_{p-1}(1) & \cdots & a_{p-1}(p-1) & 0 \\ 0 & a_{p-1}(p-1) & \cdots & a_{p-1}(1) & 1 \end{bmatrix}$$

- If $A_p(z)$ is stable, we know from Slide 16 that $|a_p(p)| < 1$, i.e. $|\rho_p| < 1$.

The Schur-Cohn stability test

- After that, we apply "z" to the 2nd row and obtain

$$\begin{bmatrix} 1 & a_{p-1}(1) & \cdots & a_{p-1}(p-1) & 0 \\ a_{p-1}(p-1) & \cdots & a_{p-1}(1) & 1 & 0 \end{bmatrix} \sim \begin{bmatrix} A_{p-1}(z) \\ A^R_{p-1}(z) \end{bmatrix}$$

  It is seen that the sequences have reduced (order $p-1$), and correspond to an allpass filter $G_{p-1}(z)$. If we had $|\rho_p| \geq 1$, i.e., $a_p(p) \geq 1$, we know $G_p(z)$ is not stable.

  Equivalently, $G_p(z)$ is stable if $|\rho_p| < 1$ and $G_{p-1}(z)$ is a stable allpass.

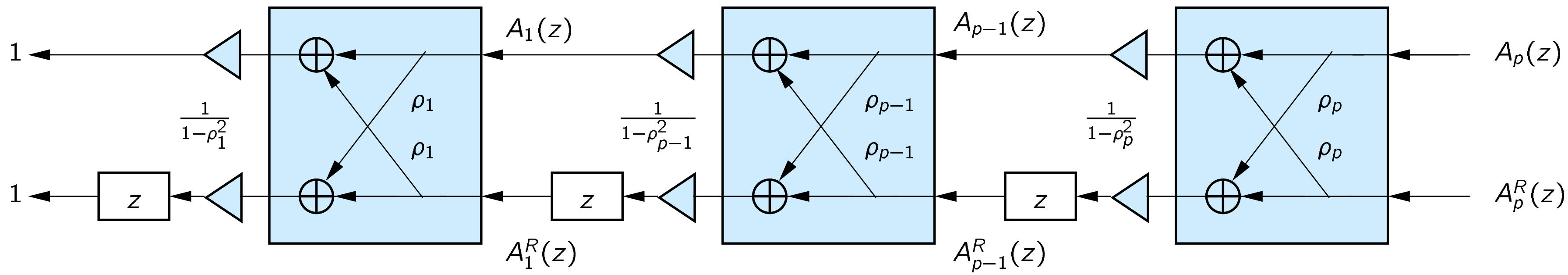
- To continue, we set $\rho_{p-1} = a_{p-1}(p-1)$, check that $|\rho_{p-1}| < 1$, apply the lattice operation, shift, etc. In the end we obtain:

$$\begin{bmatrix} A_0(z) \\ A^R_0(z) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

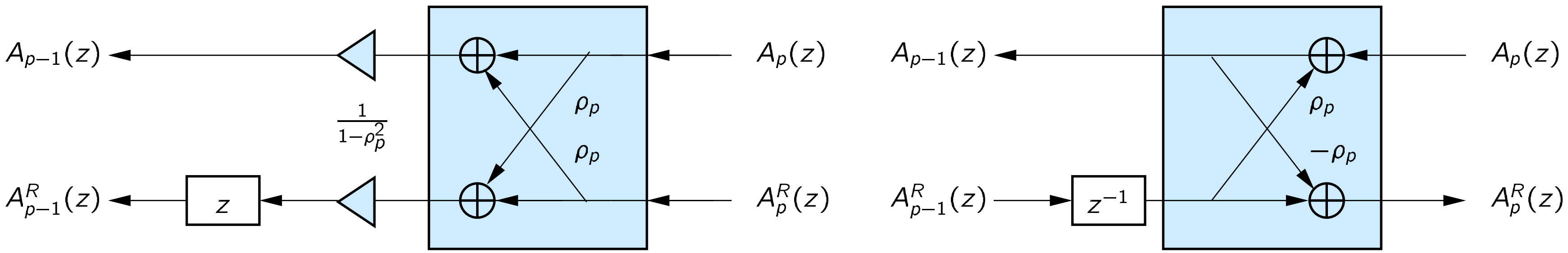
- $G_p(z)$ is stable if all $|\rho_k| < 1$.

The Schur-Cohn stability test

- At the same time, we have obtained a parametrization of the filter in terms of reflection coefficients:

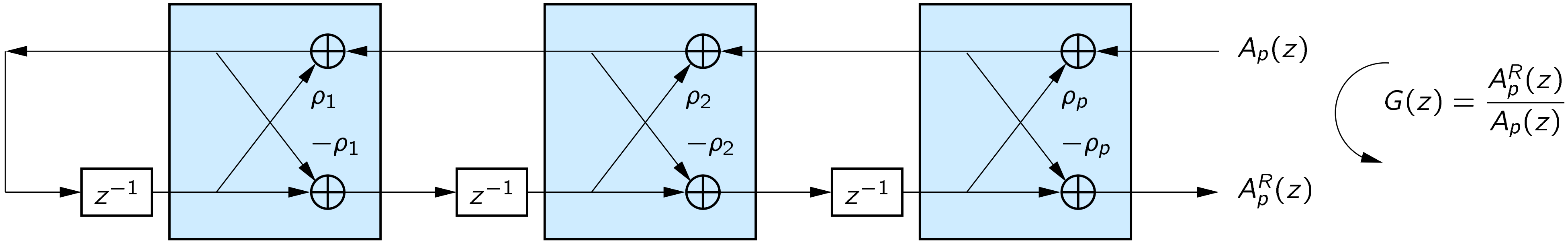


- Now use the equivalence:



The Schur-Cohn stability test

- We have found an implementation of a stable allpass filter:



This filter structure is preferred over a direct implementation due to its superior numerical properties (guaranteed stability as long as $|\rho_k| < 1$; small changes in reflection coefficients give only small changes in the pole locations).

- The given recursion to find $\{\rho_1, \cdots, \rho_p\}$ from $\{a(1), \cdots a(p)\}$ is called the Levinson "Step-Down Recursion".