Computing the covariance function of an ARMA process

Introduction

The problem of computing the covariance function for a given multivariable ARMA process is often encountered in estimation, filtering, stochastic control and communications.

Consider the ARMA process

$$A(z)\,y(t) = B(z)\,e(t), \qquad t \in \mathbf{Z}$$

where z is the shift operator defined by $zy(t) = y(t+1)$. A and B are square polynomial matrices in z with possibly complex-valued coefficients. The random sequence e is white noise so that

$$Ee(t)e^{H}(t) = \begin{cases} 0 & \text{for } t \neq s \\ I & \text{for } t = s \end{cases}$$

The superscript H indicates the complex conjugate transpose. A is assumed to be monic (that is, its leading coefficient matrix is nonsingular) with all its roots strictly inside the unit circle. Under these assumptions the ARMA process y is well-defined and asymptotically stationary.

The covariance matrix function that is to be found is defined by

$$r(\tau) = \lim_{t\to\infty} Ey(t+\tau)\,y^{H}(t)$$

Algorithm

The covariance matrix function may be computed by inverse z-transformation of the spectral density matrix

$$\Phi(z) = A^{-1}(z)C(z)C^{H}(1/z)\left(A^{-1}(1/z)\right)^{H}$$

The computation of the covariance function (Söderström, Jezek and Kucera, 1997) follows by partial fraction expansion of the spectral density in the form

$$\Phi(z) = A^{-1}(z)\,X(z) + X^{H}(1/z)\left(A^{-1}(1/z)\right)^{H}$$

This partial fraction expansion is equivalent to solving the symmetric two-sided polynomial matrix equation

$$C(z)C^{H}(1/z) = X(z)A^{H}(1/z) + A(z)X^{H}(1/z)$$

for the polynomial matrix X. Once X is available we may expand

$$A^{-1}(z)X(z) = \sum_{\tau=0}^{\infty} \hat{r}(\tau) z^{-\tau}$$

by long polynomial division. Inspection of the right-hand side shows that

$$r(\tau) = \begin{cases} \hat{r}(\tau) & \text{for } \tau > 0 \\ \hat{r}(0) + \hat{r}^H(0) & \text{for } \tau = 0 \\ \hat{r}^H(-\tau) & \text{for } \tau < 0 \end{cases}$$

Example 1: A scalar process

To illustrate the procedure first consider a scalar ARMA process y given by

$$A(z) = 1 - 2.4z + 1.43z^2$$
$$C(z) = 1$$

We develop a Polynomial Toolbox function called `covf`, which takes the polynomial matrices A and C as input arguments and has the desired covariance function r as output. Because a macro with the same name exists in the System Identification Toolbox we need to *overload* this function. Practically this means that the macro is placed in the `pol` subdirectory of the main Polynomial Toolbox directory. When MATLAB detects that `covf` is called with one or more polynomial objects as input argument then it uses the version of `covf` that is located in this subdirectory.

The macro covf

The first lines of the macro are

```
% covf

%

% This function computes the covariance function

% of the discrete-time ARMA process y defined by

%

% A(z)y(t) = C(z)e(t)

%

% with e standard white noise.
```

```
function r = covf(A,C,n)
```

The third input argument n is the number of time shifts over which the covariance function is required.

Normally at this point each Polynomial Toolbox function performs a number of correctness checks on the input arguments. We dispense with these for the purpose of this demo.

To solve the symmetric polynomial equation

$$X(z)A^H(1/z) + A(z)X^H(1/z) = C(z)C^H(1/z)$$

we use the Toolbox function `xaaxb.` Only one line is needed:

```
% Solve the two-sided polynomial matrix equation

X = xaaxb(A,C*C');
```

For the example at hand the intermediate solution at this point is

```
X =

    -20 + 8.3z + 28z^2
```

From the given polynomial A and the computed polynomial X the desired covariance function is recovered by "long division" of $X^{-1}A$. For this purpose the macro `longldiv` is available. Given the square polynomial matrix D and the polynomial matrix N this function finds the first $n + d + 1$ terms of the Laurent series expansion

$$D^{-1}(z)N(z) = Q_n z^n + Q_{n-1}z^{n-1} + \cdots + Q_1 z + R_0 + R_1 z^{-1} + R_2 z^{-2} + \cdots + R_d z^{-1} + \cdots$$

If the fraction $D^{-1}N$ is proper then the macro returns the first $d+1$ terms (with d an input parameter) of the expansion

$$D^{-1}(z)N(z) = R_0 + R_1 z^{-1} + R_2 z^{-2} + \cdots + R_d z^{-1} + \cdots$$

This is exactly what we need. Thus, the appropriate command is

```
% Apply long division to X\A

[Q,R] = longldiv(X,A,n);
```

R is returned as a polynomial matrix in the variable z^{-1} of degree n. For convenience we also return the desired covariance function as a polynomial in this variable:

```
% Construct the covariance function
r = R;
r{0} = r{0}+r{0}';
```

The macro is now complete and we may apply it to the example:

```
r = covf(1-2.4*z+1.43*z^2,1,40);
```

The resulting covariance function r may be plotted using standard MATLAB commands:

```
plot(0:40,r{:},'o')
title('Covariance function - Example 1')
xlabel('tau'), ylabel('r'), grid on
```

Fig. 1 shows the plot of the covariance function.

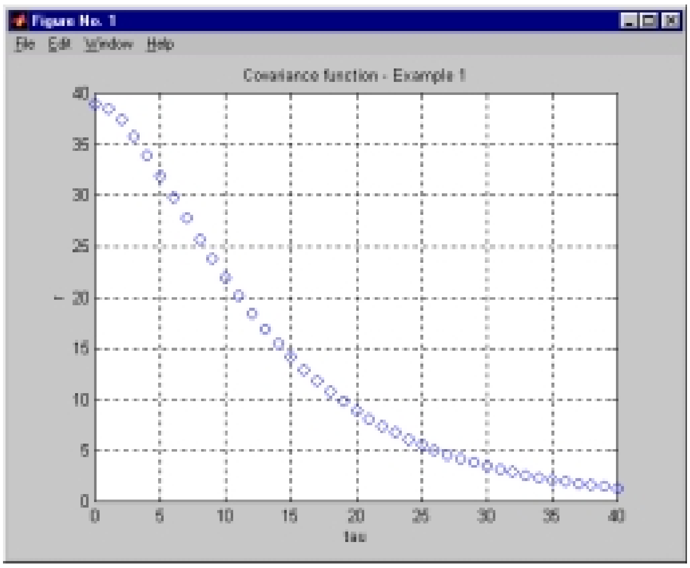

Fig. 1. Covariance function of the ARMA process of Example 1

Example 2: Two-variable ARMA process

As another example consider the computation of the covariance matrix function of the two-variable ARMA process y defined by

$$A(z)=\begin{bmatrix}1-2z & 0\\ 6 & 1-2.5z\end{bmatrix},\quad C(z)=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

As our newly created function is ready for multivariable processes we may use it as it is after defining the data:

```
  A = [1-2*z 0; 6 1-2.5*z]; C = eye(2,2);
r = covf(A,C,10);
```

The output r now is a 2×2 polynomial matrix of degree 10. The coefficients $r\{i\}$, $i = 1, 2, \cdots, 10$, constitute the desired covariance function. They may be plotted in a single frame by the following sequence of standard MATLAB commands that we include at the end of the macro `covf`:

```
% Plot the covariance function

figure; clf

k = length(C);

for i = 1:k

   for j = 1:k

      subplot(k,k,(i-1)*k+j)

      plot(0:n,r{:}(i,j),'o')

      grid on; xlabel('tau')

   end

end
```

The resulting plot is shown in Fig. 2. The subplot in position (i, j) shows the scalar covariance function $r_{ij}(\tau)=\lim_{\tau\to\infty}\operatorname{cov}[y_i(t+\tau),y_j(t)]$.

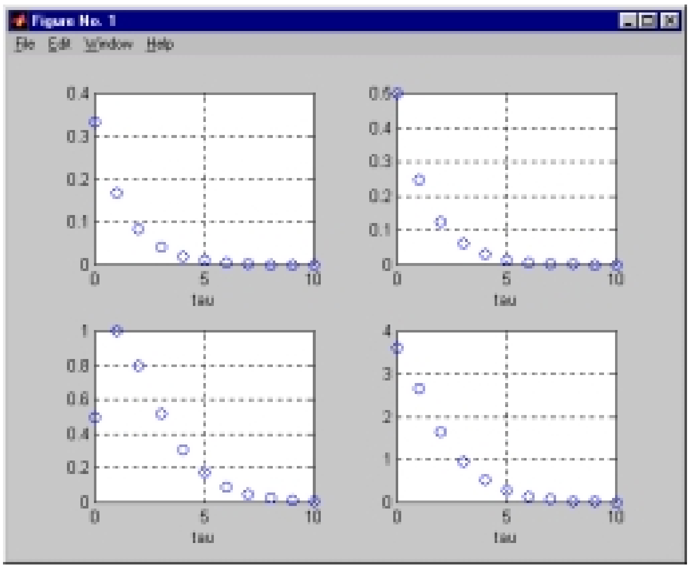

Fig. 2. Covariance function of the ARMA process of Example 2