Classical Filtering With Linear Algebra

Jesse Perla, Thomas J. Sargent and John Stachurski

December 4, 2020

1 Contents

2 Overview

This is a sequel to the earlier lecture Classical Control with Linear Algebra.

That lecture used linear algebra – in particular, the LU decomposition – to formulate and solve a class of linear-quadratic optimal control problems.

In this lecture, we'll be using a closely related decomposition, the Cholesky decomposition , to solve linear prediction and filtering problems.

We exploit the useful fact that there is an intimate connection between two superficially different classes of problems:

- deterministic linear-quadratic (LQ) optimal control problems
- linear least squares prediction and filtering problems

The first class of problems involves no randomness, while the second is all about randomness.

Nevertheless, essentially the same mathematics solves both type of problem.

This connection, which is often termed "duality," is present whether one uses "classical" or "recursive" solution procedures.

In fact we saw duality at work earlier when we formulated control and prediction problems recursively in lectures LQ dynamic programming problems, A first look at the Kalman filter, and The permanent income model.

A useful consequence of duality is that

- With every LQ control problem there is implicitly affiliated a linear least squares prediction or filtering problem.
- With every linear least squares prediction or filtering problem there is implicitly affiliated a LQ control problem.

An understanding of these connections has repeatedly proved useful in cracking interesting applied problems.

For example, Sargent [6] [chs. IX, XIV] and Hansen and Sargent [2] formulated and solved control and filtering problems using z-transform methods.

In this lecture we investigate these ideas using mostly elementary linear algebra.

2.1 References

Useful references include [7], [2], [4], [1], and [3].

2.2 Setup

```
In [1]: using InstantiateFromURL
        # optionally add arguments to force installation: instantiate = true,
      ↪precompile = true
        github_project("QuantEcon/quantecon-notebooks-julia", version = "0.8.0")
```

```
In [2]: using LinearAlgebra, Statistics
```

3 Infinite Horizon Prediction and Filtering Problems

We pose two related prediction and filtering problems.

We let Y_t be a univariate m^{th} order moving average, covariance stationary stochastic process,

$$Y_t = d(L) u_t \tag{1}$$

where $d(L) = \sum_{j=0}^{m} d_j L^j$, and u_t is a serially uncorrelated stationary random process satisfying

$$\begin{aligned} \mathbb{E} u_t &= 0 \\ \mathbb{E} u_t u_s &= \begin{cases} 1 & \text{if } t = s \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

We impose no conditions on the zeros of $d(z)$.

A second covariance stationary process is X_t given by

$$X_t = Y_t + \varepsilon_t \tag{3}$$

where ε_t is a serially uncorrelated stationary random process with $\mathbb{E}\varepsilon_t = 0$ and $\mathbb{E}\varepsilon_t \varepsilon_s = 0$ for all distinct t and s.

We also assume that $\mathbb{E}\varepsilon_t u_s = 0$ for all t and s.

The **linear least squares prediction problem** is to find the L_2 random variable $\hat{X}_{t+j}$ among linear combinations of $\{X_t,\ X_{t-1}, ...\}$ that minimizes $\mathbb{E}(\hat{X}_{t+j} - X_{t+j})^2$.

That is, the problem is to find a $\gamma_j(L) = \sum_{k=0}^{\infty} \gamma_{jk} L^k$ such that $\sum_{k=0}^{\infty} |\gamma_{jk}|^2 < \infty$ and $\mathbb{E}[\gamma_j(L)X_t - X_{t+j}]^2$ is minimized.

The **linear least squares filtering problem** is to find a $b(L) = \sum_{j=0}^{\infty} b_j L^j$ such that $\sum_{j=0}^{\infty} |b_j|^2 < \infty$ and $\mathbb{E}[b(L)X_t - Y_t]^2$ is minimized.

Interesting versions of these problems related to the permanent income theory were studied by [3].

3.1 Problem formulation

These problems are solved as follows.

The covariograms of Y and X and their cross covariogram are, respectively,

$$\begin{aligned} C_X(\tau) &= \mathbb{E}X_t X_{t-\tau} \\ C_Y(\tau) &= \mathbb{E}Y_t Y_{t-\tau} \qquad \tau = 0, \pm 1, \pm 2, ... \\ C_{Y,X}(\tau) &= \mathbb{E}Y_t X_{t-\tau} \end{aligned} \tag{4}$$

The covariance and cross covariance generating functions are defined as

$$\begin{aligned} g_X(z) &= \sum_{\tau=-\infty}^{\infty} C_X(\tau) z^\tau \\ g_Y(z) &= \sum_{\tau=-\infty}^{\infty} C_Y(\tau) z^\tau \\ g_{YX}(z) &= \sum_{\tau=-\infty}^{\infty} C_{YX}(\tau) z^\tau \end{aligned} \tag{5}$$

The generating functions can be computed by using the following facts.

Let v_{1t} and v_{2t} be two mutually and serially uncorrelated white noises with unit variances.

That is, $\mathbb{E}v_{1t}^2 = \mathbb{E}v_{2t}^2 = 1, \mathbb{E}v_{1t} = \mathbb{E}v_{2t} = 0, \mathbb{E}v_{1t}v_{2s} = 0$ for all t and s, $\mathbb{E}v_{1t}v_{1t-j} = \mathbb{E}v_{2t}v_{2t-j} = 0$ for all $j \neq 0$.

Let x_t and y_t be two random process given by

$$\begin{aligned} y_t &= A(L)v_{1t} + B(L)v_{2t} \\ x_t &= C(L)v_{1t} + D(L)v_{2t} \end{aligned}$$

Then, as shown for example in [6] [ch. XI], it is true that

$$\begin{aligned} g_y(z) &= A(z)A(z^{-1}) + B(z)B(z^{-1}) \\ g_x(z) &= C(z)C(z^{-1}) + D(z)D(z^{-1}) \\ g_{yx}(z) &= A(z)C(z^{-1}) + B(z)D(z^{-1}) \end{aligned} \tag{6}$$

Applying these formulas to (1) – (4), we have

$$\begin{aligned} g_Y(z) &= d(z)d(z^{-1}) \\ g_X(z) &= d(z)d(z^{-1}) + h \\ g_{YX}(z) &= d(z)d(z^{-1}) \end{aligned} \tag{7}$$

The key step in obtaining solutions to our problems is to factor the covariance generating function $g_X(z)$ of X.

The solutions of our problems are given by formulas due to Wiener and Kolmogorov.

These formulas utilize the Wold moving average representation of the X_t process,

$$X_t = c\left(L\right)\eta_t \tag{8}$$

where $c(L) = \sum_{j=0}^{m} c_j\, L^j$, with

$$c_0\eta_t = X_t - \hat{\mathbb{E}}[X_t|X_{t-1}, X_{t-2}, ...] \tag{9}$$

Here $\hat{\mathbb{E}}$ is the linear least squares projection operator.

Equation (9) is the condition that $c_0\eta_t$ can be the one-step ahead error in predicting X_t from its own past values.

Condition (9) requires that η_t lie in the closed linear space spanned by $[X_t,\ X_{t-1}, ...]$.

This will be true if and only if the zeros of $c(z)$ do not lie inside the unit circle.

It is an implication of (9) that η_t is a serially uncorrelated random process, and that a normalization can be imposed so that $\mathbb{E}\eta_t^2 = 1$.

Consequently, an implication of (8) is that the covariance generating function of X_t can be expressed as

$$g_X(z) = c\left(z\right)c\left(z^{-1}\right) \tag{10}$$

It remains to discuss how $c(L)$ is to be computed.

Combining (6) and (10) gives

$$d(z)\, d(z^{-1}) + h = c\left(z\right)c\left(z^{-1}\right) \tag{11}$$

Therefore, we have already showed constructively how to factor the covariance generating function $g_X(z) = d(z)\, d\left(z^{-1}\right) + h$.

We now introduce the **annihilation operator**:

$$\left[\sum_{j=-\infty}^{\infty} f_j\, L^j\right]_+ \equiv \sum_{j=0}^{\infty} f_j\, L^j \tag{12}$$

In words, $[\quad]_+$ means "ignore negative powers of L".

We have defined the solution of the prediction problem as $\hat{\mathbb{E}}[X_{t+j}|X_t,\ X_{t-1}, ...] = \gamma_j\left(L\right)X_t$.

Assuming that the roots of $c(z) = 0$ all lie outside the unit circle, the Wiener-Kolmogorov formula for $\gamma_j(L)$ holds:

$$\gamma_j\left(L\right) = \left[\frac{c(L)}{L^j}\right]_+ c\left(L\right)^{-1} \tag{13}$$

We have defined the solution of the filtering problem as $\hat{\mathbb{E}}[Y_t \mid X_t, X_{t-1}, ...] = b(L)X_t$.

The Wiener-Kolomogorov formula for $b(L)$ is

$$b(L) = \left(\frac{g_{YX}(L)}{c(L^{-1})} \right)_+ c(L)^{-1}$$

or

$$b(L) = \left[\frac{d(L)d(L^{-1})}{c(L^{-1})} \right]_+ c(L)^{-1} \tag{14}$$

Formulas (13) and (14) are discussed in detail in [8] and [6].

The interested reader can there find several examples of the use of these formulas in economics Some classic examples using these formulas are due to [3].

As an example of the usefulness of formula (14), we let X_t be a stochastic process with Wold moving average representation

$$X_t = c(L)\eta_t$$

where $\mathbb{E}\eta_t^2 = 1,$ and $c_0\eta_t = X_t - \hat{\mathbb{E}}[X_t|X_{t-1}, ...], c(L) = \sum_{j=0}^m c_j L.$

Suppose that at time t, we wish to predict a geometric sum of future X's, namely

$$y_t \equiv \sum_{j=0}^{\infty} \delta^j X_{t+j} = \frac{1}{1 - \delta L^{-1}} X_t$$

given knowledge of $X_t, X_{t-1}, ...$

We shall use (14) to obtain the answer.

Using the standard formulas (6), we have that

$$\begin{aligned} g_{yx}(z) &= (1 - \delta z^{-1})c(z)c(z^{-1}) \\ g_x(z) &= c(z)c(z^{-1}) \end{aligned}$$

Then (14) becomes

$$b(L) = \left[\frac{c(L)}{1 - \delta L^{-1}} \right]_+ c(L)^{-1} \tag{15}$$

In order to evaluate the term in the annihilation operator, we use the following result from [2].

Proposition Let

- $g(z) = \sum_{j=0}^{\infty} g_j z^j$ where $\sum_{j=0}^{\infty} |g_j|^2 < +\infty$
- $h\left(z^{-1}\right) = \left(1 - \delta_1 z^{-1}\right) ... \left(1 - \delta_n z^{-1}\right)$, where $|\delta_j| < 1$, for $j = 1, ..., n$

Then

$$\left[\frac{g(z)}{h(z^{-1})}\right]_+ = \frac{g(z)}{h(z^{-1})} - \sum_{j=1}^{n} \frac{\delta_j g(\delta_j)}{\prod_{\substack{k=1\\k\neq j}}^{n}(\delta_j - \delta_k)} \left(\frac{1}{z-\delta_j}\right) \tag{16}$$

and, alternatively,

$$\left[\frac{g(z)}{h(z^{-1})}\right]_+ = \sum_{j=1}^{n} B_j \left(\frac{z g(z) - \delta_j g(\delta_j)}{z-\delta_j}\right) \tag{17}$$

where $B_j = 1/\prod_{\substack{k=1\\k+j}}^{n}(1-\delta_k/\delta_j)$.

Applying formula (17) of the proposition to evaluating (15) with $g(z) = c(z)$ and $h(z^{-1}) = 1-\delta z^{-1}$ gives

$$b(L) = \left[\frac{Lc(L) - \delta c(\delta)}{L-\delta}\right] c(L)^{-1}$$

or

$$b(L) = \left[\frac{1 - \delta c(\delta) L^{-1} c(L)^{-1}}{1-\delta L^{-1}}\right]$$

Thus, we have

$$\hat{\mathbb{E}}\left[\sum_{j=0}^{\infty} \delta^j X_{t+j} | X_t,\, x_{t-1}, ...\right] = \left[\frac{1 - \delta c(\delta) L^{-1} c(L)^{-1}}{1-\delta L^{-1}}\right] X_t \tag{18}$$

This formula is useful in solving stochastic versions of problem 1 of lecture Classical Control with Linear Algebra in which the randomness emerges because $\{a_t\}$ is a stochastic process.

The problem is to maximize

$$\mathbb{E}_0 \lim_{N\to\infty} \sum_{t-0}^{N} \beta^t \left[a_t\, y_t - \frac{1}{2}\, h y_t^2 - \frac{1}{2}\, [d(L) y_t]^2\right] \tag{19}$$

where $\mathbb{E}_t$ is mathematical expectation conditioned on information known at t, and where $\{a_t\}$ is a covariance stationary stochastic process with Wold moving average representation

$$a_t = c(L)\, \eta_t$$

where

$$c(L) = \sum_{j=0}^{\tilde{n}} c_j L^j$$

and

$$\eta_t = a_t - \hat{\mathbb{E}}[a_t | a_{t-1}, ...]$$

The problem is to maximize (19) with respect to a contingency plan expressing y_t as a function of information known at t, which is assumed to be $(y_{t-1},\ y_{t-2}, \dots, a_t,\ a_{t-1}, ...)$.

The solution of this problem can be achieved in two steps.

First, ignoring the uncertainty, we can solve the problem assuming that $\{a_t\}$ is a known sequence.

The solution is, from above,

$$c(L)y_t = c(\beta L^{-1})^{-1} a_t$$

or

$$(1 - \lambda_1 L) \dots (1 - \lambda_m L) y_t = \sum_{j=1}^{m} A_j \sum_{k=0}^{\infty} (\lambda_j \beta)^k \, a_{t+k} \tag{20}$$

Second, the solution of the problem under uncertainty is obtained by replacing the terms on the right-hand side of the above expressions with their linear least squares predictors.

Using (18) and (20), we have the following solution

$$(1 - \lambda_1 L) \dots (1 - \lambda_m L) y_t = \sum_{j=1}^{m} A_j \left[\frac{1 - \beta\lambda_j \, c(\beta\lambda_j) L^{-1} c(L)^{-1}}{1 - \beta\lambda_j L^{-1}} \right] a_t$$

4 Finite Dimensional Prediction

Let $(x_1, x_2, \dots, x_T)' = x$ be a $T \times 1$ vector of random variables with mean $\mathbb{E}x = 0$ and covariance matrix $\mathbb{E}xx' = V$.

Here V is a $T \times T$ positive definite matrix.

We shall regard the random variables as being ordered in time, so that x_t is thought of as the value of some economic variable at time t.

For example, x_t could be generated by the random process described by the Wold representation presented in equation (8).

In this case, V_{ij} is given by the coefficient on $z^{|i-j|}$ in the expansion of $g_x(z) = d(z)\, d(z^{-1}) + h$, which equals $h + \sum_{k=0}^{\infty} d_k d_{k+|i-j|}$.

We shall be interested in constructing j step ahead linear least squares predictors of the form

$$\hat{\mathbb{E}}\left[x_T | x_{T-j}, x_{T-j+1}, \dots, x_1\right]$$

where $\hat{\mathbb{E}}$ is the linear least squares projection operator.

The solution of this problem can be exhibited by first constructing an orthonormal basis of random variables ε for x.

Since V is a positive definite and symmetric, we know that there exists a (Cholesky) decomposition of V such that

$$V = L^{-1}(L^{-1})'$$

or

$$L\,V\,L' = I$$

where L is lower-trangular, and therefore so is L^{-1}.

Form the random variable $Lx = \varepsilon$.

Then ε is an orthonormal basis for x, since L is nonsingular, and $\mathbb{E}\,\varepsilon\,\varepsilon' = L\mathbb{E}xx'L' = I$.

It is convenient to write out the equations $Lx = \varepsilon$ and $L^{-1}\varepsilon = x$

$$\begin{aligned} L_{11}x_1 &= \varepsilon_1 \\ L_{21}x_1 + L_{22}x_2 &= \varepsilon_2 \\ &\vdots \\ L_{T1}\,x_1\ \ldots + L_{TTx_T} &= \varepsilon_T \end{aligned} \tag{21}$$

or

$$\sum_{j=0}^{t-1} L_{t,t-j}\,x_{t-j} = \varepsilon_t, \quad t = 1,\, 2, \ldots T \tag{22}$$

We also have

$$x_t = \sum_{j=0}^{t-1} L^{-1}_{t,t-j}\,\varepsilon_{t-j}\ . \tag{23}$$

Notice from (23) that x_t is in the space spanned by ε_t, $\varepsilon_{t-1}, \ldots, \varepsilon_1$, and from (22) that ε_t is in the space spanned by x_t, $x_{t-1}, \ldots,\, x_1$.

Therefore, we have that for $t-1 \geq m \geq 1$

$$\hat{\mathbb{E}}[x_t \mid x_{t-m},\, x_{t-m-1}, \ldots, x_1] = \hat{\mathbb{E}}[x_t \mid \varepsilon_{t-m}, \varepsilon_{t-m-1}, \ldots, \varepsilon_1] \tag{24}$$

For $t-1 \geq m \geq 1$ rewrite (23) as

$$x_t = \sum_{j=0}^{m-1} L^{-1}_{t,t-j}\,\varepsilon_{t-j} + \sum_{j=m}^{t-1} L^{-1}_{t,t-j}\,\varepsilon_{t-j} \tag{25}$$

Representation (25) is an orthogonal decomposition of x_t into a part $\sum_{j=m}^{t-1} L^{-1}_{t,t-j}\,\varepsilon_{t-j}$ that lies in the space spanned by $[x_{t-m},\, x_{t-m+1},\, \ldots, x_1]$, and an orthogonal component not in this space.

4.1 Implementation

Code that computes solutions to LQ control and filtering problems using the methods described here and in Classical Control with Linear Algebra can be found in the file control_and_filter.jl.

Here's how it looks

In [3]:
```
using Polynomials.PolyCompat, LinearAlgebra
import Polynomials.PolyCompat: roots, coeffs

function LQFilter(d, h, y_m;
                  r = nothing,
                  β = nothing,
                  h_eps = nothing)

    m = length(d) - 1
    m == length(y_m) ||
        throw(ArgumentError("y_m and d must be of same length = $m"))

    # define the coefficients of ⊠ up front
    ϕ = zeros(2m + 1)
    for i in -m:m
        ϕ[m-i+1] = sum(diag(d*d', -i))
    end
    ϕ[m+1] = ϕ[m+1] + h

    # if r is given calculate the vector ⊠_r
    if isnothing(r)
        k = nothing
        ϕ_r = nothing
    else
        k = size(r, 1) - 1
        ϕ_r = zeros(2k + 1)

        for i = -k:k
            ϕ_r[k-i+1] = sum(diag(r*r', -i))
        end

        if isnothing(h_eps) == false
            ϕ_r[k+1] = ϕ_r[k+1] + h_eps
        end
    end

    # if β is given, define the transformed variables
    if isnothing(β)
        β = 1.0
    else
        d = β.^(collect(0:m)/2) * d
        y_m = y_m * β.^(- collect(1:m)/2)
    end

    return (d = d, h = h, y_m = y_m, m = m, ϕ = ϕ, β = β,
            ϕ_r = ϕ_r, k = k)
end

function construct_W_and_Wm(lqf, N)
```

```
    d, m = lqf.d, lqf.m

    W = zeros(N + 1, N + 1)
    W_m = zeros(N + 1, m)

    # terminal conditions
    D_m1 = zeros(m + 1, m + 1)
    M = zeros(m + 1, m)

    # (1) constuct the D_{m+1} matrix using the formula

    for j in 1:(m+1)
        for k in j:(m+1)
            D_m1[j, k] = dot(d[1:j, 1], d[k-j+1:k, 1])
        end
    end

    # Make the matrix symmetric
    D_m1 = D_m1 + D_m1' - Diagonal(diag(D_m1))

    # (2) Construct the M matrix using the entries of D_m1

    for j in 1:m
        for i in (j + 1):(m + 1)
            M[i, j] = D_m1[i-j, m+1]
        end
    end
    M

    # Euler equations for t = 0, 1, ..., N-(m+1)
    ϕ, h = lqf.ϕ, lqf.h

    W[1:(m + 1), 1:(m + 1)] = D_m1 + h * I
    W[1:(m + 1), (m + 2):(2m + 1)] = M

    for (i, row) in enumerate((m + 2):(N + 1 - m))
        W[row, (i + 1):(2m + 1 + i)] = ϕ'
    end

    for i in 1:m
        W[N - m + i + 1 , end-(2m + 1 - i)+1:end] = ϕ[1:end-i]
    end

    for i in 1:m
        W_m[N - i + 2, 1:(m - i)+1] = ϕ[(m + 1 + i):end]
    end

    return W, W_m
end

function roots_of_characteristic(lqf)
    m, ϕ = lqf.m, lqf.ϕ

    # Calculate the roots of the 2m-polynomial
    ϕ_poly = Poly(ϕ[end:-1:1])
    proots = roots(ϕ_poly)
    # sort the roots according to their length (in descending order)
    roots_sorted = sort(proots, by=abs)[end:-1:1]
```

```
        z_0 = sum(ϕ) / polyval(poly(proots), 1.0)
        z_1_to_m = roots_sorted[1:m]     # we need only those outside the unit
↪circle
        λ = 1 ./ z_1_to_m
        return z_1_to_m, z_0, λ
    end

    function coeffs_of_c(lqf)
        m = lqf.m
        z_1_to_m, z_0, λ = roots_of_characteristic(lqf)
        c_0 = (z_0 * prod(z_1_to_m) * (-1.0)^m)^(0.5)
        c_coeffs = coeffs(poly(z_1_to_m)) * z_0 / c_0
        return c_coeffs
    end

    function solution(lqf)
        z_1_to_m, z_0, λ = roots_of_characteristic(lqf)
        c_0 = coeffs_of_c(lqf)[end]
        A = zeros(lqf.m)
        for j in 1:m
            denom = 1 - λ/λ[j]
            A[j] = c_0^(-2) / prod(denom[1:m .!= j])
        end
        return λ, A
    end

    function construct_V(lqf; N = nothing)
        if isnothing(N)
            error("N must be provided!!")
        end
        if !(N isa Integer)
            throw(ArgumentError("N must be Integer!"))
        end

        ϕ_r, k = lqf.ϕ_r, lqf.k
        V = zeros(N, N)
        for i in 1:N
            for j in 1:N
                if abs(i-j) ≤ k
                    V[i, j] = ϕ_r[k + abs(i-j)+1]
                end
            end
        end
        return V
    end

    function simulate_a(lqf, N)
        V = construct_V(N + 1)
        d = MVNSampler(zeros(N + 1), V)
        return rand(d)
    end

    function predict(lqf, a_hist, t)
        N = length(a_hist) - 1
        V = construct_V(N + 1)

        aux_matrix = zeros(N + 1, N + 1)
        aux_matrix[1:t+1 , 1:t+1 ] = Matrix(I, t + 1, t + 1)
```

```
        L = cholesky(V).U'
        Ea_hist = inv(L) * aux_matrix * L * a_hist

        return Ea_hist
    end

    function optimal_y(lqf, a_hist, t = nothing)
        β, y_m, m = lqf.β, lqf.y_m, lqf.m

        N = length(a_hist) - 1
        W, W_m = construct_W_and_Wm(lqf, N)

        F = lu(W, Val(true))

        L, U = F
        D = diagm(0 => 1.0 ./ diag(U))
        U = D * U
        L = L * diagm(0 => 1.0 ./ diag(D))

        J = reverse(Matrix(I, N + 1, N + 1), dims = 2)

        if isnothing(t)                      # if the problem is deterministic
            a_hist = J * a_hist

            # transform the a sequence if β is given
            if β != 1
                a_hist =  reshape(a_hist * (β^(collect(N:0)/ 2)), N + 1, 1)
            end

            ā = a_hist - W_m * y_m          # ā from the lecutre
            Uy = \(L, ā)                    # U @ ȳ = L^{-1}ā from the lecture
            ȳ = \(U, Uy)                    # ȳ = U^{-1}L^{-1}ā
            # Reverse the order of ȳ with the matrix J
            J = reverse(Matrix(I, N + m + 1, N + m + 1), dims = 2)
            y_hist = J * vcat(ȳ, y_m)       # y_hist : concatenated y_m and ȳ
            # transform the optimal sequence back if β is given
            if β != 1
                y_hist = y_hist .* β.^(- collect(-m:N)/2)
            end

        else                                 # if the problem is stochastic and
↪we look at
        it
            Ea_hist = reshape(predict(a_hist, t), N + 1, 1)
            Ea_hist = J * Ea_hist

            ā = Ea_hist - W_m * y_m         # ā from the lecutre
            Uy = \(L, ā)                    # U @ ȳ = L^{-1}ā from the lecture
            ȳ = \(U, Uy)                    # ȳ = U^{-1}L^{-1}ā

            # Reverse the order of ȳ with the matrix J
            J = reverse(Matrix(I, N + m + 1, N + m + 1), dims = 2)
            y_hist = J * vcat(ȳ, y_m)       # y_hist : concatenated y_m and ȳ
        end
```

```
        return y_hist, L, U, y
    end
```

```
Out[3]: optimal_y (generic function with 2 methods)
```

Let's use this code to tackle two interesting examples.

4.2 Example 1

Consider a stochastic process with moving average representation

$$x_t = (1 - 2L)\varepsilon_t$$

where ε_t is a serially uncorrelated random process with mean zero and variance unity.

We want to use the Wiener-Kolmogorov formula (13) to compute the linear least squares forecasts $\mathbb{E}[x_{t+j} \mid x_t, x_{t-1}, ...]$, for $j = 1, 2$.

We can do everything we want by setting $d = r$, generating an instance of LQFilter, then invoking pertinent methods of LQFilter

```
In [4]: m = 1
        y_m = zeros(m)
        d = [1.0, -2.0]
        r = [1.0, -2.0]
        h = 0.0
        example = LQFilter(d, h, y_m, r=d)
```

```
Out[4]: (d = [1.0, -2.0], h = 0.0, y_m = [0.0], m = 1, ϕ = [-2.0, 5.0, -2.0], β =
        ↪1.0, ϕ_r =
        [-2.0, 5.0, -2.0], k = 1)
```

The Wold representation is computed by example.coefficients_of_c().

Let's check that it "flips roots" as required

```
In [5]: coeffs_of_c(example)
```

```
Out[5]: 2-element Array{Float64,1}:
          2.0
         -1.0
```

```
In [6]: roots_of_characteristic(example)
```

```
Out[6]: ([2.0], -2.0, [0.5])
```

Now let's form the covariance matrix of a time series vector of length N and put it in V.

Then we'll take a Cholesky decomposition of $V = L^{-1}L^{-1} = LiLi'$ and use it to form the vector of "moving average representations" $x = Li\varepsilon$ and the vector of "autoregressive representations" $Lx = \varepsilon$

```
In [7]: V = construct_V(example,N=5)
```

```
Out[7]: 5×5 Array{Float64,2}:
          5.0  -2.0   0.0   0.0   0.0
         -2.0   5.0  -2.0   0.0   0.0
          0.0  -2.0   5.0  -2.0   0.0
          0.0   0.0  -2.0   5.0  -2.0
          0.0   0.0   0.0  -2.0   5.0
```

Notice how the lower rows of the "moving average representations" are converging to the appropriate infinite history Wold representation

```
In [8]: F = cholesky(V)
        Li = F.L
```

```
Out[8]: 5×5 LowerTriangular{Float64,Array{Float64,2}}:
          2.23607     □          □          □          □
         -0.894427   2.04939     □          □          □
          0.0       -0.9759     2.01187     □          □
          0.0        0.0       -0.9941     2.00294     □
          0.0        0.0        0.0       -0.998533   2.00073
```

Notice how the lower rows of the "autoregressive representations" are converging to the appropriate infinite history autoregressive representation

```
In [9]: L = inv(Li)
```

```
Out[9]: 5×5 LowerTriangular{Float64,Array{Float64,2}}:
          0.447214    □          □          □          □
          0.19518    0.48795     □          □          □
          0.0946762  0.236691   0.49705     □          □
          0.0469898  0.117474   0.246696   0.499266    □
          0.0234518  0.0586295  0.123122   0.249176   0.499817
```

Remark Let $\pi(z) = \sum_{j=0}^{m} \pi_j z^j$ and let $z_1, \ldots, z_k$ be the zeros of $\pi(z)$ that are inside the unit circle, $k < m$.

Then define

$$\theta(z) = \pi(z)\left(\frac{(z_1 z - 1)}{(z - z_1)}\right)\left(\frac{(z_2 z - 1)}{(z - z_2)}\right)\ldots\left(\frac{(z_k z - 1)}{(z - z_k)}\right)$$

The term multiplying $\pi(z)$ is termed a "Blaschke factor".

Then it can be proved directly that

$$\theta(z^{-1})\theta(z) = \pi(z^{-1})\pi(z)$$

and that the zeros of $\theta(z)$ are not inside the unit circle.

4.3 Example 2

Consider a stochastic process X_t with moving average representation

$$X_t = (1 - \sqrt{2}L^2)\varepsilon_t$$

where ε_t is a serially uncorrelated random process with mean zero and variance unity.

Let's find a Wold moving average representation for x_t.

Let's use the Wiener-Kolomogorov formula (13) to compute the linear least squares forecasts $\hat{\mathbb{E}}\left[X_{t+j} \mid X_{t-1}, ...\right]$ for $j = 1, 2, 3$.

We proceed in the same way as example 1

```
In [10]: m = 2
         y_m = [0.0, 0.0]
         d = [1, 0, -sqrt(2)]
         r = [1, 0, -sqrt(2)]
         h = 0.0
         example = LQFilter(d, h, y_m, r = d)
```

```
Out[10]: (d = [1.0, 0.0, -1.4142135623730951], h = 0.0, y_m = [0.0, 0.0], m = 2, ϕ =
         [-1.4142135623730951, 0.0, 3.0000000000000004, 0.0, -1.4142135623730951],
  ↪β = 1.0, ϕ_r =
         [-1.4142135623730951, 0.0, 3.0000000000000004, 0.0, -1.4142135623730951],
  ↪k = 2)
```

```
In [11]: coeffs_of_c(example)
```

```
Out[11]: 3-element Array{Float64,1}:
           1.4142135623731003
          -0.0
          -1.0000000000000064
```

```
In [12]: roots_of_characteristic(example)
```

```
Out[12]: ([1.1892071150027195, -1.1892071150027195], -1.4142135623731096, [0.
  ↪8408964152537157,
         -0.8408964152537157])
```

```
In [13]: V = construct_V(example, N = 8)
```

```
Out[13]: 8×8 Array{Float64,2}:
           3.0       0.0      -1.41421   0.0       …   0.0       0.0       0.0
           0.0       3.0       0.0      -1.41421       0.0       0.0       0.0
          -1.41421   0.0       3.0       0.0           0.0       0.0       0.0
           0.0      -1.41421   0.0       3.0          -1.41421   0.0       0.0
           0.0       0.0      -1.41421   0.0           0.0      -1.41421   0.0
           0.0       0.0       0.0      -1.41421   …   3.0       0.0      -1.41421
           0.0       0.0       0.0       0.0           0.0       3.0       0.0
           0.0       0.0       0.0       0.0          -1.41421   0.0       3.0
```

```
In [14]: F = cholesky(V)
         Li = F.L
         Li[end-2:end, :]
```

```
Out[14]: 3×8 Array{Float64,2}:
          0.0  0.0  0.0  -0.92582   0.0        1.46385   0.0      0.0
          0.0  0.0  0.0   0.0      -0.966092   0.0       1.43759  0.0
          0.0  0.0  0.0   0.0       0.0       -0.966092  0.0      1.43759
```

```
In [15]: L = inv(Li)
```

```
Out[15]: 8×8 LowerTriangular{Float64,Array{Float64,2}}:
          0.57735    ⋅         ⋅         ⋅         …   ⋅          ⋅          ⋅
          0.0        0.57735   ⋅         ⋅             ⋅          ⋅          ⋅
          0.308607   0.0       0.654654  ⋅             ⋅          ⋅          ⋅
          0.0        0.308607  0.0       0.654654      ⋅          ⋅          ⋅
          0.19518    0.0       0.414039  0.0           ⋅          ⋅          ⋅
          0.0        0.19518   0.0       0.414039  …   0.68313    ⋅          ⋅
          0.131165   0.0       0.278243  0.0           0.0        0.695608   ⋅
          0.0        0.131165  0.0       0.278243      0.459078   0.0        0.695608
```

4.4 Prediction

It immediately follows from the "orthogonality principle" of least squares (see [1] or [6] [ch. X]) that

$$\begin{aligned}\hat{\mathbb{E}}[x_t \mid x_{t-m},\, x_{t-m+1}, \dots x_1] &= \sum_{j=m}^{t-1} L^{-1}_{t,t-j}\, \varepsilon_{t-j} \\ &= [L^{-1}_{t,1}\, L^{-1}_{t,2},\, \dots,\, L^{-1}_{t,t-m}\; 0\; 0 \dots 0] L\, x\end{aligned} \tag{26}$$

This can be interpreted as a finite-dimensional version of the Wiener-Kolmogorov m-step ahead prediction formula.

We can use (26) to represent the linear least squares projection of the vector x conditioned on the first s observations $[x_s, x_{s-1} \dots, x_1]$.

We have

$$\hat{\mathbb{E}}[x \mid x_s, x_{s-1}, \dots, x_1] = L^{-1} \begin{bmatrix} I_s & 0 \\ 0 & 0_{(t-s)} \end{bmatrix} Lx \tag{27}$$

This formula will be convenient in representing the solution of control problems under uncertainty.

Equation (23) can be recognized as a finite dimensional version of a moving average representation.

Equation (22) can be viewed as a finite dimension version of an autoregressive representation.

Notice that even if the x_t process is covariance stationary, so that V is such that V_{ij} depends only on $|i - j|$, the coefficients in the moving average representation are time-dependent, there being a different moving average for each t.

If x_t is a covariance stationary process, the last row of L^{-1} converges to the coefficients in the Wold moving average representation for $\{x_t\}$ as $T \to \infty$.

Further, if x_t is covariance stationary, for fixed k and $j > 0$, $L^{-1}_{T,T-j}$ converges to $L^{-1}_{T-k,T-k-j}$ as $T \to \infty$.

That is, the "bottom" rows of L^{-1} converge to each other and to the Wold moving average coefficients as $T \to \infty$.

This last observation gives one simple and widely-used practical way of forming a finite T approximation to a Wold moving average representation.

First, form the covariance matrix $\mathbb{E}xx' = V$, then obtain the Cholesky decomposition $L^{-1}L^{-1'}$ of V, which can be accomplished quickly on a computer.

The last row of L^{-1} gives the approximate Wold moving average coefficients.

This method can readily be generalized to multivariate systems.

5 Combined Finite Dimensional Control and Prediction

Consider the finite-dimensional control problem, maximize

$$\mathbb{E}\sum_{t=0}^{N}\left\{a_t y_t - \frac{1}{2}hy_t^2 - \frac{1}{2}[d(L)y_t]^2\right\}, \qquad h > 0$$

where $d(L) = d_0 + d_1 L + ... + d_m L^m$, L is the lag operator, $\bar{a} = [a_N, a_{N-1} \dots, a_1, a_0]'$ a random vector with mean zero and $\mathbb{E}\,\bar{a}\bar{a}' = V$.

The variables $y_{-1}, \dots, y_{-m}$ are given.

Maximization is over choices of $y_0, y_1 \dots, y_N$, where y_t is required to be a linear function of $\{y_{t-s-1}, t+m-1 \geq 0;\ a_{t-s}, t \geq s \geq 0\}$.

We saw in the lecture Classical Control with Linear Algebra that the solution of this problem under certainty could be represented in feedback-feedforward form

$$U\bar{y} = L^{-1}\bar{a} + K\begin{bmatrix} y_{-1} \\ \vdots \\ y_{-m} \end{bmatrix}$$

for some $(N+1) \times m$ matrix K.

Using a version of formula (26), we can express $\hat{\mathbb{E}}[\bar{a} \mid a_s, a_{s-1}, \dots, a_0]$ as

$$\hat{\mathbb{E}}[\bar{a} \mid a_s, a_{s-1}, \dots, a_0] = \tilde{U}^{-1}\begin{bmatrix} 0 & 0 \\ 0 & I_{(s+1)} \end{bmatrix}\tilde{U}\bar{a}$$

where $I_{(s+1)}$ is the $(s+1) \times (s+1)$ identity matrix, and $V = \tilde{U}^{-1}\tilde{U}^{-1'}$, where $\tilde{U}$ is the *upper* trangular Cholesky factor of the covariance matrix V.

(We have reversed the time axis in dating the a's relative to earlier)

The time axis can be reversed in representation (27) by replacing L with L^T.

The optimal decision rule to use at time $0 \leq t \leq N$ is then given by the $(N-t+1)^{\text{th}}$ row of

$$U\bar{y} = L^{-1}\tilde{U}^{-1}\begin{bmatrix} 0 & 0 \\ 0 & I_{(t+1)} \end{bmatrix}\tilde{U}\bar{a} + K\begin{bmatrix} y_{-1} \\ \vdots \\ y_{-m} \end{bmatrix}$$

6 Exercises

6.1 Exercise 1

Let $Y_t = (1-2L)u_t$ where u_t is a mean zero white noise with $\mathbb{E}u_t^2 = 1$. Let

$$X_t = Y_t + \varepsilon_t$$

where ε_t is a serially uncorrelated white noise with $\mathbb{E}\varepsilon_t^2 = 9$, and $\mathbb{E}\varepsilon_t u_s = 0$ for all t and s.

Find the Wold moving average representation for X_t.

Find a formula for the A_{1j}'s in

$$\mathbb{E}\widehat{X}_{t+1} \mid X_t, X_{t-1}, ... = \sum_{j=0}^{\infty} A_{1j}X_{t-j}$$

Find a formula for the A_{2j}'s in

$$\hat{\mathbb{E}}X_{t+2} \mid X_t, X_{t-1}, ... = \sum_{j=0}^{\infty} A_{2j}X_{t-j}$$

6.2 Exercise 2

(Multivariable Prediction) Let Y_t be an $(n \times 1)$ vector stochastic process with moving average representation

$$Y_t = D(L)U_t$$

where $D(L) = \sum_{j=0}^{m} D_j L^J, D_j$ an $n \times n$ matrix, U_t an $(n \times 1)$ vector white noise with :math: mathbb{E} U_t =0 for all t, $\mathbb{E}U_t U_s' = 0$ for all $s \neq t$, and $\mathbb{E}U_t U_t' = I$ for all t.

Let ε_t be an $n \times 1$ vector white noise with mean 0 and contemporaneous covariance matrix H, where H is a positive definite matrix.

Let $X_t = Y_t + \varepsilon_t$.

Define the covariograms as $C_X(\tau) = \mathbb{E}X_t X_{t-\tau}', C_Y(\tau) = \mathbb{E}Y_t Y_{t-\tau}', C_{YX}(\tau) = \mathbb{E}Y_t X_{t-\tau}'$.

Then define the matrix covariance generating function, as in (21), only interpret all the objects in (21) as matrices.

Show that the covariance generating functions are given by

$$
\begin{aligned}
g_y(z) &= D(z)D(z^{-1})' \\
g_X(z) &= D(z)D(z^{-1})' + H \\
g_{YX}(z) &= D(z)D(z^{-1})'
\end{aligned}
$$

A factorization of $g_X(z)$ can be found (see [5] or [8]) of the form

$$
D(z)D(z^{-1})' + H = C(z)C(z^{-1})', \quad C(z) = \sum_{j=0}^{m} C_j z^j
$$

where the zeros of $|C(z)|$ do not lie inside the unit circle.

A vector Wold moving average representation of X_t is then

$$
X_t = C(L)\eta_t
$$

where η_t is an $(n \times 1)$ vector white noise that is "fundamental" for X_t.

That is, $X_t - \hat{\mathbb{E}}\left[X_t \mid X_{t-1}, X_{t-2} \ldots\right] = C_0\, \eta_t$.

The optimum predictor of X_{t+j} is

$$
\hat{\mathbb{E}}\left[X_{t+j} \mid X_t, X_{t-1}, \ldots\right] = \left[\frac{C(L)}{L^j}\right]_+ \eta_t
$$

If $C(L)$ is invertible, i.e., if the zeros of $\det C(z)$ lie strictly outside the unit circle, then this formula can be written

$$
\hat{\mathbb{E}}\left[X_{t+j} \mid X_t, X_{t-1}, \ldots\right] = \left[\frac{C(L)}{L^J}\right]_+ C(L)^{-1} X_t
$$

References

[1] Papoulis Athanasios and S Unnikrishna Pillai. *Probability, random variables, and stochastic processes*. Mc-Graw Hill, 1991.

[2] Lars Peter Hansen and Thomas J Sargent. Formulating and estimating dynamic linear rational expectations models. *Journal of Economic Dynamics and control*, 2:7–46, 1980.

[3] John F Muth. Optimal properties of exponentially weighted forecasts. *Journal of the american statistical association*, 55(290):299–306, 1960.

[4] Sophocles J Orfanidis. *Optimum Signal Processing: An Introduction*. McGraw Hill Publishing, New York, New York, 1988.

[5] Y. A. Rozanov. *Stationary Random Processes*. Holden-Day, San Francisco, 1967.

[6] Thomas J Sargent. *Macroeconomic Theory*. Academic Press, New York, 2nd edition, 1987.

[7] Peter Whittle. *Prediction and regulation by linear least-square methods*. English Univ. Press, 1963.

[8] Peter Whittle. *Prediction and Regulation by Linear Least Squares Methods*. University of Minnesota Press, Minneapolis, Minnesota, 2nd edition, 1983.