Classical Prediction and Filtering With Linear Algebra

Thomas J. Sargent and John Stachurski

September 22, 2020

1 Contents

- Overview 2
- Finite Dimensional Prediction 3
- Combined Finite Dimensional Control and Prediction 4
- Infinite Horizon Prediction and Filtering Problems 5
- Exercises 6

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 types 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 begin to investigate these ideas by using mostly elementary linear algebra. This is the main purpose and focus of the lecture.

However, after showing matrix algebra formulas, we'll summarize classic infinite-horizon formulas built on z-transform and lag operator methods.

And we'll occasionally refer to some of these formulas from the infinite dimensional problems as we present the finite time formulas and associated linear algebra.

We'll start with the following standard import:

```python
In [1]: import numpy as np
```

### 2.1 References

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

### 3 Finite Dimensional Prediction

Let \((x_1, x_2, \ldots, x_T)’ = x\) 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.

The \(i, j\) component \(Ex_i x_j\) of \(V\) is the **inner product** between \(x_i\) and \(x_j\).

We regard the random variables as being ordered in time so that \(x_i\) is thought of as the value of some economic variable at time \(t\).

For example, \(x_i\) could be generated by the random process described by the Wold representation presented in equation (16) in the section below on infinite dimensional prediction and filtering.

In that 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 want to construct \(j\) step ahead linear least squares predictors of the form

\[
\hat{\mathbb{E}} \left[ x_T \mid x_{T-j}, x_{T-j+1}, \ldots, x_1 \right]
\]

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

(Sometimes \(\hat{\mathbb{E}}\) is called the wide-sense expectations operator)

To find linear least squares predictors it is helpful first to construct a \(T \times 1\) vector \(\varepsilon\) of random variables that form an orthonormal basis for the vector of random variables \(x\).

The key insight here comes from noting that because the covariance matrix \(V\) of \(x\) is a positive definite and symmetric, there exists a (Cholesky) decomposition of \(V\) such that
\[ V = L^{-1}(L^{-1})' \]

and

\[ LVL' = I \]

where \( L \) and \( L^{-1} \) are both lower triangular.

Form the \( T \times 1 \) random vector \( \varepsilon = Lx \).

The random vector \( \varepsilon \) is an orthonormal basis for \( x \) because

- \( L \) is nonsingular
- \( \mathbb{E} \varepsilon \varepsilon' = L \mathbb{E} xx' L' = I \)
- \( x = L^{-1} \varepsilon \)

It is enlightening to write out and interpret the equations \( Lx = \varepsilon \) and \( L^{-1} \varepsilon = x \).

First, we’ll write \( Lx = \varepsilon \)

\[
\begin{align*}
L_{11}x_1 &= \varepsilon_1 \\
L_{21}x_1 + L_{22}x_2 &= \varepsilon_2 \\
\vdots \\
L_{T1}x_1 + \cdots + L_{TT}x_T &= \varepsilon_T
\end{align*}
\]

or

\[ \sum_{j=0}^{t-1} L_{t,t-j} x_{t-j} = \varepsilon_t, \quad t = 1, 2, \ldots, T \]

Next, we write \( L^{-1} \varepsilon = x \)

\[
\begin{align*}
x_1 &= L_{11}^{-1} \varepsilon_1 \\
x_2 &= L_{22}^{-1} \varepsilon_2 + L_{21}^{-1} \varepsilon_1 \\
\vdots \\
x_T &= L_{TT}^{-1} \varepsilon_T + L_{T,T-1}^{-1} \varepsilon_{T-1} + \cdots + L_{T,1}^{-1} \varepsilon_1
\end{align*}
\]

or

\[ x_t = \sum_{j=0}^{t-1} L_{t,t-j}^{-1} \varepsilon_{t-j} \]

where \( L_{i,j}^{-1} \) denotes the \( i, j \) element of \( L^{-1} \).

From (2), it follows that \( \varepsilon_t \) is in the linear subspace spanned by \( x_t, x_{t-1}, \ldots, x_1 \).

From (4) it follows that that \( x_t \) is in the linear subspace spanned by \( \varepsilon_t, \varepsilon_{t-1}, \ldots, \varepsilon_1 \).

Equation (2) forms a sequence of autoregressions that for \( t = 1, \ldots, T \) express \( x_t \) as linear functions of \( x_s, s = 1, \ldots, t-1 \) and a random variable \((L_{t,t})^{-1} \varepsilon_t\) that is orthogonal to each component of \( x_s, s = 1, \ldots, t-1 \).
(Here $(L_{t,t})^{-1}$ denotes the reciprocal of $L_{t,t}$ while $L_{t,t}^{-1}$ denotes the $t,t$ element of $L^{-1}$).

The equivalence of the subspaces spanned by $\varepsilon_t, \ldots, \varepsilon_1$ and $x_t, \ldots, x_1$ means that for $t - 1 \geq m \geq 1$

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

To proceed, it is useful to drill down and note that for $t - 1 \geq m \geq 1$ we can rewrite (4) in the form of the **moving average representation**

$$x_t = \sum_{j=0}^{m-1} L_{t,t-j}^{-1} \varepsilon_{t-j} + \sum_{j=m}^{t-1} L_{t,t-j}^{-1} \varepsilon_{t-j}$$

(6)

Representation (6) is an orthogonal decomposition of $x_t$ into a part $\sum_{j=m}^{t-1} L_{t,t-j}^{-1} \varepsilon_{t-j}$ that lies in the space spanned by $[x_{t-m}, x_{t-m+1}, \ldots, x_1]$ and an orthogonal component $\sum_{j=m}^{t-1} L_{t,t-j}^{-1} \varepsilon_{t-j}$ that does not lie in that space but instead in a linear space known as its **orthogonal complement**.

It follows that

$$\hat{E}[x_t \mid x_{t-m}, x_{t-m-1}, \ldots, x_1] = \sum_{j=0}^{m-1} L_{t,t-j}^{-1} \varepsilon_{t-j}$$

3.1 Implementation

Here’s the code that computes solutions to LQ control and filtering problems using the methods described here and in :doc: lu_tricks.

```python
import numpy as np
import scipy.stats as spst
import scipy.linalg as sla

class LQFilter:
    def __init__(self, d, h, y_m, r=None, h_eps=None, β=None):
        ...
```

**Parameters**

*d* : list or numpy.array (1-D or a 2-D column vector)

The order of the coefficients: $[d_0, d_1, \ldots, d_m]$

*h* : scalar

Parameter of the objective function (corresponding to the quadratic term)

*y_m* : list or numpy.array (1-D or a 2-D column vector)

Initial conditions for $y$

*r* : list or numpy.array (1-D or a 2-D column vector)

The order of the coefficients: $[r_0, r_1, \ldots, r_k]$

(optional, if not defined -> deterministic problem)

*β* : scalar

Discount factor (optional, default value is one)
self.h = h
self.d = np.asarray(d)
self.m = self.d.shape[0] - 1
self.y_m = np.asarray(y_m)

if self.m == self.y_m.shape[0]:
    self.y_m = self.y_m.reshape(self.m, 1)
else:
    raise ValueError("y_m must be of length m = {self.m:d}")

#---------------------------------------------
# Define the coefficients of \phi upfront
#---------------------------------------------
ϕ = np.zeros(2 * self.m + 1)
for i in range(-self.m, self.m + 1):
    ϕ[self.m - i] = np.sum(np.diag(self.d.reshape(self.m + 1, 1) \  
                                @ self.d.reshape(1, self.m + 1),
                              k=-1)

ϕ[self.m] = ϕ[self.m] + self.h
self.ϕ = ϕ

#---------------------------------------------
# If r is given calculate the vector \phi_r
#---------------------------------------------
if r is None:
    pass
else:
    self.r = np.asarray(r)
    self.k = self.r.shape[0] - 1
    ϕ_r = np.zeros(2 * self.k + 1)
    for i in range(-self.k, self.k + 1):
        ϕ_r[self.k - i] = np.sum(np.diag(self.r.reshape(self.k + 1, \  
                                            -1) \  
                                @ self.r.reshape(1, self.k + 1),
                              k=-1)

    if h_eps is None:
        self.ϕ_r = ϕ_r
    else:
        ϕ_r[self.k] = ϕ_r[self.k] + h_eps
        self.ϕ_r = ϕ_r

#---------------------------------------------
# If \beta is given, define the transformed variables
#---------------------------------------------
if β is None:
    self.β = 1
else:
    self.β = β
    self.d = self.β**(np.arange(self.m + 1)/2) * self.d
    self.y_m = self.y_m * (self.β**((np.arange(1, self.m + 1)/2)) \  
                          .reshape(self.m, 1)
```python
def construct_W_and_Wm(self, N):
    """
    This constructs the matrices W and W_m for a given number of periods N
    """
    m = self.m
    d = self.d
    W = np.zeros((N + 1, N + 1))
    W_m = np.zeros((N + 1, m))
    #---------------------------------------
    # Terminal conditions
    #---------------------------------------
    D_m1 = np.zeros((m + 1, m + 1))
    M = np.zeros((m + 1, m))
    # (1) Construct the D_{m+1} matrix using the formula
    for j in range(m + 1):
        for k in range(j, m + 1):
            D_m1[j, k] = d[:j + 1] @ d[k - j: k + 1]
    # Make the matrix symmetric
    D_m1 = D_m1 + D_m1.T - np.diag(np.diag(D_m1))
    # (2) Construct the M matrix using the entries of D_m1
    for j in range(m):
        for i in range(j + 1, m + 1):
            M[i, j] = D_m1[i - j - 1, m]
    #----------------------------------------------
    # Euler equations for t = 0, 1, ..., N-(m+1)
    #----------------------------------------------
    ϕ = self.ϕ
    W[:(m + 1), :(m + 1)] = D_m1 + self.h * np.eye(m + 1)
    W[:(m + 1), (m + 1):(2 * m + 1)] = M
    for i, row in enumerate(np.arange(m + 1, N + 1 - m)):
        W[row, (i + 1):(2 * m + 2 + i)] = ϕ
    for i in range(1, m + 1):
        W[N - m + i, -(2 * m + 1 - i):] = ϕ[:-i]
    for i in range(m):
        W_m[N - i, :(m - i)] = ϕ[(m + 1 + i):]
    return W, W_m
def roots_of_characteristic(self):
    """
    This function calculates z_0 and the 2m roots of the characteristic
    equation associated with the Euler equation (1.7)
    """
    Note:
```

numpy.poly1d(roots, True) defines a polynomial using its roots that can be evaluated at any point. If $x_1, x_2, \ldots, x_m$ are the roots then $p(x) = (x - x_1)(x - x_2)\ldots(x - x_m)$

```python
m = self.m
ϕ = self.ϕ

# Calculate the roots of the 2m-polynomial
roots = np.roots(ϕ)
# Sort the roots according to their length (in descending order)
roots_sorted = roots[np.argsort(abs(roots))[:-1]]

z_0 = ϕ.sum() / np.poly1d(roots, True)(1)
z_1_to_m = roots_sorted[:m]  # We need only those outside the unit circle

λ = 1 / z_1_to_m
return z_1_to_m, z_0, λ
```

def coeffs_of_c(self):
    '''
    This function computes the coefficients $\{c_j, j = 0, 1, \ldots, m\}$ for $c(z) = \sum_{j = 0}^{m} c_j z^j$
    
    Based on the expression (1.9). The order is $c_coeffs = [c_0, c_1, \ldots, c_{m-1}, c_m]$
    '''
    z_1_to_m, z_0 = self.roots_of_characteristic()[:2]
    c_0 = (z_0 * np.prod(z_1_to_m).real * (-1)**self.m)**(.5)
    c_coeffs = np.poly1d(z_1_to_m, True).c * z_0 / c_0
    return c_coeffs[:-1]

def solution(self):
    '''
    This function calculates $\{λ_j, j = 1, \ldots, m\}$ and $\{A_j, j = 1, \ldots, m\}$ of the expression (1.15)
    '''
    λ = self.roots_of_characteristic()[2]
c_0 = self.coeffs_of_c()[-1]
A = np.zeros(self.m, dtype=complex)
for j in range(self.m):
    denom = 1 - λ / λ[j]
    A[j] = c_0**(-2) / np.prod(denom[np.arange(self.m) != j])

return λ, A

def construct_V(self, N):
    '''
    This function constructs the covariance matrix for $x^N$ (see section 6)
    '''
    for a given period $N$
V = np.zeros((N, N))
ϕ_r = self.ϕ_r

for i in range(N):
    for j in range(N):
        if abs(i-j) <= self.k:
            V[i, j] = ϕ_r[self.k + abs(i-j)]

return V

def simulate_a(self, N):
    """
    Assuming that the u's are normal, this method draws a random path
    for x^N
    """
    V = self.construct_V(N + 1)
    d = spst.multivariate_normal(np.zeros(N + 1), V)
    return d.rvs()

def predict(self, a_hist, t):
    """
    This function implements the prediction formula discussed in
    section 6 (1.59)
    It takes a realization for a^N, and the period in which the
    prediction is
    Output: E[abar | a_t, a_{t-1}, ..., a_1, a_0]
    """
    N = np.asarray(a_hist).shape[0] - 1
    a_hist = np.asarray(a_hist).reshape(N + 1, 1)
    V = self.construct_V(N + 1)

    aux_matrix = np.zeros((N + 1, N + 1))
    aux_matrix[:,(t + 1), :].T = np.eye(t + 1)
    L = la.cholesky(V).T
    Ea_hist = la.inv(L) @ aux_matrix @ L @ a_hist
    return Ea_hist

def optimal_y(self, a_hist, t=None):
    """
    - if t is NOT given it takes a_hist (list or numpy.array) as a
deterministic a_t
    - if t is given, it solves the combined control prediction problem
    (section 7)(by default, t == None -> deterministic)

    for a given sequence of a_t (either deterministic or a particular
realization), it calculates the optimal y_t sequence using the method
of the lecture

    Note:
    """
    scipy.linalg.lu normalizes L, U so that L has unit diagonal elements
    To make things consistent with the lecture, we need an auxiliary
diagonal matrix D which renormalizes L and U

    """
N = np.asarray(a_hist).shape[0] - 1
W, W_m = self.construct_W_and_Wm(N)
L, U = la.lu(W, permute_l=True)
D = np.diag(1 / np.diag(U))
U = D @ U
L = L @ np.diag(1 / np.diag(D))
J = np.fliplr(np.eye(N + 1))

if t is None:  # If the problem is deterministic
    a_hist = J @ np.asarray(a_hist).reshape(N + 1, 1)
    a_bar = a_hist - W_m @ self.y_m  # a_bar from the lecture
    Uy = np.linalg.solve(L, a_bar)  # U @ y_bar = L^{-1}
    y_bar = np.linalg.solve(U, Uy)  # y_bar = U^{-1}L^{-1}

    J = np.fliplr(np.eye(N + self.m + 1))  # y_hist : concatenated y_m and y_bar
    y_hist = J @ np.vstack([y_bar, self.y_m])

    if self.β != 1:
        y_hist = y_hist * (self.β**(np.arange(N + 1) / 2))[:,::-1] \
            .reshape(N + 1, 1)

return y_hist, L, U, y_bar

else:  # If the problem is stochastic and we look at it
    Ea_hist = self.predict(a_hist, t).reshape(N + 1, 1)
    Ea_hist = J @ Ea_hist
    a_bar = Ea_hist - W_m @ self.y_m  # a_bar from the lecture
    Uy = np.linalg.solve(L, a_bar)  # U @ y_bar = L^{-1}
    y_bar = np.linalg.solve(U, Uy)  # y_bar = U^{-1}L^{-1}

    J = np.fliplr(np.eye(N + self.m + 1))  # y_hist : concatenated y_m and y_bar
    y_hist = J @ np.vstack([y_bar, self.y_m])

    return y_hist, L, U, y_bar

Let’s use this code to tackle two interesting examples.
3.2 Example 1

Consider a stochastic process with moving average representation

\[ x_t = (1 - 2L) \varepsilon_t \]

where \( \varepsilon_t \) is a serially uncorrelated random process with mean zero and variance unity.

If we were to use the tools associated with infinite dimensional prediction and filtering to be described below, we would use the Wiener-Kolmogorov formula (21) to compute the linear least squares forecasts \( \mathbb{E}[x_{t+j} \mid x_t, x_{t-1}, \ldots] \), for \( j = 1, 2 \).

But we can do everything we want by instead using our finite dimensional tools and setting \( d = r \), generating an instance of LQFilter, then invoking pertinent methods of LQFilter.

\[
\text{In [3]}: \quad m = 1 \\
y_m = \text{np.asarray([0.])}.\text{reshape(m, 1)} \\
d = \text{np.asarray([1, -2])} \\
r = \text{np.asarray([1, -2])} \\
h = 0.0 \\
\text{example} = \text{LQFilter}(d, h, y_m, r=d)
\]

The Wold representation is computed by \text{example.coefficients_of_c}().

Let’s check that it “flips roots” as required

\[
\text{In [4]}: \quad \text{example.coeffs_of_c}() \\
\text{Out[4]}: \quad \text{array([ 2., -1.])}
\]

\[
\text{In [5]}: \quad \text{example.roots_of_characteristic}() \\
\text{Out[5]}: \quad (\text{array([2.]), -2.0, \text{array([0.5])})}
\]

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 \( V = L^{-1}L^{-1} \) and use it to form the vector of “moving average representations” \( x = L^{-1} \varepsilon \) and the vector of “autoregressive representations” \( Lx = \varepsilon \).

\[
\text{In [6]}: \quad V = \text{example.construct_V(N=5)} \\
\quad \text{print}(V) \\
\quad \quad \left[
\begin{array}{cccc}
5. & -2. & 0. & 0. \\
-2. & 5. & -2. & 0. \\
0. & -2. & 5. & -2. \\
0. & 0. & -2. & 5. \\
0. & 0. & 0. & -2. \\
\end{array}\right]
\]

Notice how the lower rows of the “moving average representations” are converging to the appropriate infinite history Wold representation to be described below when we study infinite horizon-prediction and filtering.
In [7]: `Li = np.linalg.cholesky(V)
   print(Li)

   [[ 2.23606798 0. 0. 0. 0. ]
   [-0.89442719 2.04939015 0. 0. 0. ]
   [ 0. -0.97590007 2.01186954 0. 0. ]
   [ 0. 0. -0.99410024 2.00293902 0. ]
   [ 0. 0. 0. -0.99853265 2.000733 ]]]

Notice how the lower rows of the “autoregressive representations” are converging to the appropriate infinite-history autoregressive representation to be described below when we study infinite horizon-prediction and filtering

In [8]: `L = np.linalg.inv(Li)
   print(L)

   [[0.4472136 0. 0. 0. 0. ]
   [0.19518001 0.48795004 0. 0. 0. ]
   [0.09467621 0.23669053 0.49705012 0. 0. ]
   [0.04698977 0.11747443 0.2466963 0.49926632 0. ]
   [0.02345182 0.05862954 0.12312203 0.24917554 0.49981682]]

3.3 Example 2

Consider a stochastic process $X_t$ with moving average representation

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

where $\varepsilon_t$ is a serially uncorrelated random process with mean zero and variance unity.

Let’s find a Wold moving average representation for $x_t$ that will prevail in the infinite-history context to be studied in detail below.

To do this, we’ll use the Wiener-Kolomogorov formula (21) presented below to compute the linear least squares forecasts $\hat{E}[X_{t+j} | X_{t-1},...]$ for $j = 1, 2, 3$.

We proceed in the same way as in example 1

In [9]: `m = 2
   y_m = np.asarray([.0,.0]).reshape(m, 1)
   d = np.asarray([[1,0,-np.sqrt(2)])
   r = np.asarray([[1,0,-np.sqrt(2)])
   h = 0.0
   example = LQFilter(d, h, y_m, r=d)
   example.coeffs_of_c()

Out[9]: array([ 1.41421356, -0. , -1. ])

In [10]: example.roots_of_characteristic()
In [11]: V = example.construct_V(N=8)
    print(V)

    
[[ 3.  0. -1.41421356  0.      0.      0.      ]
 [ 0.  3.  0. -1.41421356  0.      0.      ]
 [-1.41421356  0.  3.  0.      -1.41421356  0.      ]
 [ 0. -1.41421356  0.  3.      0.      -1.41421356  0.      ]
 [ 0.  0.  -1.41421356  0.      3.      0.      ]
 [-1.41421356  0.  0.  3.      0.      -1.41421356  0.      ]
 [ 0.  0.  0.   0.      0.      -1.41421356  0.      ]
 [ 0.  0.  0.   0.      0.      0.      -1.41421356  0.      ]
 [ 0.  0.  0.   0.      0.      0.      0.      -1.41421356  0.      ]

In [12]: Li = np.linalg.cholesky(V)
    print(Li[-3:, :])

    
[[ 0.     0.     0.     -0.9258201  0.      1.46385011 ]
 [ 0.     0.     0.     0.     -0.96609178  0.      ]
 [ 0.  1.43759058  0.      0.     0.     -0.96609178  0.      ]
 [ 0.  0.     0.     0.     0.     -1.41421356  0.      ]
 [ 0.  0.     0.     0.     0.      0.     -1.41421356  0.      ]
 [ 0.  0.     0.     0.     0.     0.      0.      -1.41421356  0.      ]
 [ 0.  0.     0.     0.     0.     0.     0.      0.      1.43759058]]

In [13]: L = np.linalg.inv(Li)
    print(L)

    
[[0.57735027  0.     0.     0.     0.     0.     ]
 [0.  0.57735027  0.     0.     0.     0.     ]
 [0.  0.  0.65465367  0.     0.     0.     ]
 [0.  0.  0.  0.65465367  0.     0.     ]
 [0.  0.  0.  0.  0.68313005  0.     ]
 [0.  0.  0.  0.  0.  0.68313005  0.     ]
 [0.  0.  0.  0.  0.  0.  0.69560834  0.     ]
 [0.  0.  0.  0.  0.  0.  0.  0.69560834  0.     ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.69560834]]
3.4 Prediction

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

\[ \hat{E}[x_t | x_{t-m}, x_{t-m+1}, \ldots, x_1] = \sum_{j=m}^{t-1} L_{t,j}^{-1} \varepsilon_{t-j} \]

\[ = [L_{t,1}^{-1} L_{t,2}^{-1}, \ldots, L_{t,t-m}^{-1} 0 0 \ldots 0]L x \]

(7)

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

We can use (7) to represent the linear least squares projection of the vector \( x \) conditioned on the first \( s \) observations \( [x_s, x_{s-1}, \ldots, x_1] \).

We have

\[ \hat{E}[x | x_s, x_{s-1}, \ldots, x_1] = L^{-1} \begin{bmatrix} I_s & 0 \\ 0 & 0 \end{bmatrix}_{(t-s)} L x \]

(8)

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

Equation (4) can be recognized as a finite dimensional version of a moving average representation.

Equation (2) 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 \( j > 0 \), \( L_{T,T-j}^{-1} \) converges to \( L_{T-k,T-k-j}^{-1} \) 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.

4 Combined Finite Dimensional Control and Prediction

Consider the finite-dimensional control problem, maximize
$$\mathbb{E} \sum_{t=0}^{N} \left\{ a_t y_t - \frac{1}{2} h y_t^2 - \frac{1}{2} [d(L)y_t]^2 \right\}, \quad h > 0$$

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

The variables \( y_{-1}, \ldots, y_{-m} \) are given.

Maximization is over choices of \( y_0, y_1, \ldots, y_N \), 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 the feedback-feedforward form

\[
U \tilde{y} = L^{-1} \tilde{a} + K \begin{bmatrix} y_{-1} \\ \vdots \\ y_{-m} \end{bmatrix}
\]

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

Using a version of formula (7), we can express \( \widehat{\mathbb{E}}[\tilde{a} | a_s, a_{s-1}, \ldots, a_0] \) as

\[
\widehat{\mathbb{E}}[\tilde{a} | a_s, a_{s-1}, \ldots, a_0] = \tilde{U}^{-1} \begin{bmatrix} 0 & 0 \\ 0 & I_{(s+1)} \end{bmatrix} \tilde{U} \tilde{a}
\]

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 triangular 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 (8) by replacing \( L \) with \( L^T \).

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

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

### 5 Infinite Horizon Prediction and Filtering Problems

It is instructive to compare the finite-horizon formulas based on linear algebra decompositions of finite-dimensional covariance matrices with classic formulas for infinite horizon and infinite history prediction and control problems.

These classic infinite horizon formulas used the mathematics of \( z \)-transforms and lag operators.

We’ll meet interesting lag operator and \( z \)-transform counterparts to our finite horizon matrix formulas.

We pose two related prediction and filtering problems.

We let \( Y_t \) be a univariate \( m^{th} \) order moving average, covariance stationary stochastic process,
\[ Y_t = d(L)u_t \] (9)

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

\[
\begin{align*}
\mathbb{E} u_t &= 0 \\
\mathbb{E} u_t u_s &= \begin{cases} 1 & \text{if } t = s \\ 0 & \text{otherwise} \end{cases}
\end{align*}
\] (10)

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

A second covariance stationary process is \( X_t \) given by

\[ X_t = Y_t + \varepsilon_t \] (11)

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

We also assume that \( \mathbb{E} \varepsilon_t \varepsilon_s = 0 \) 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 \( \{X_t, X_{t-1}, \ldots\} \) 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].

### 5.1 Problem Formulation

These problems are solved as follows.

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

\[
\begin{align*}
C_X(\tau) &= \mathbb{E}X_t X_{t-\tau} \\
C_Y(\tau) &= \mathbb{E}Y_t Y_{t-\tau} \quad \tau = 0, \pm 1, \pm 2, \ldots \\
C_{Y,X}(\tau) &= \mathbb{E}Y_t X_{t-\tau}
\end{align*}
\] (12)

The covariance and cross-covariance generating functions are defined as

\[
\begin{align*}
g_X(z) &= \sum_{\tau=-\infty}^{\infty} C_X(\tau) z^{\tau} \\
g_Y(z) &= \sum_{\tau=-\infty}^{\infty} C_Y(\tau) z^{\tau} \\
g_{YX}(z) &= \sum_{\tau=-\infty}^{\infty} C_{YX}(\tau) z^{\tau}
\end{align*}
\] (13)
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, $\mathbb{E}v_{1t}^2 = \mathbb{E}v_{2t}^2 = 1, \mathbb{E}v_{1t} = \mathbb{E}v_{2t} = 0$ 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 processes given by

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

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

\begin{align*}
  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{align*}

Applying these formulas to (9) – (12), we have

\begin{align*}
  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{align*}

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,

\begin{equation}
  X_t = c(L)\eta_t
\end{equation}

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

\begin{equation}
  c_0\eta_t = X_t - \hat{\mathbb{E}}[X_t|X_{t-1}, X_{t-2}, ...]
\end{equation}

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

Equation (17) 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 (17) requires that $\eta_t$ 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 (17) that $\eta_t$ is a serially uncorrelated random process and that normalization can be imposed so that $\mathbb{E}\eta_t^2 = 1$.

Consequently, an implication of (16) is that the covariance generating function of $X_t$ can be expressed as

\begin{equation}
  g_X(z) = c(z)c(z^{-1})
\end{equation}
It remains to discuss how $c(L)$ is to be computed.

Combining (14) and (18) gives

$$d(z)d(z^{-1}) + h = c(z)c(z^{-1})$$  \hspace{1cm} (19)$$

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

We now introduce the **annihilation operator**:

$$\left[ \sum_{j=-\infty}^{\infty} f_j L^j \right]_+ \equiv \sum_{j=0}^{\infty} f_j L^j$$  \hspace{1cm} (20)$$

In words, $[\phantom{X}]+$ means “ignore negative powers of $L$”.

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

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

$$\gamma_j(L) = \left[ \frac{c(L)}{L^j} \right]_+ c(L)^{-1} \hspace{1cm} (21)$$

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

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

$$b(L) = \left[ \frac{g_{Y,X}(L)}{c(L^{-1})} \right]_+ c(L)^{-1}$$

or

$$b(L) = \left[ \frac{d(L)d(L^{-1})}{c(L^{-1})} \right]_+ c(L)^{-1} \hspace{1cm} (22)$$

Formulas (21) and (22) 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 (22), 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 $c_0\eta_t = X_t - \hat{E}[X_t | X_{t-1}, \ldots]$, $c(L) = \sum_{j=0}^{\infty} c_j L$.

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 $X_t, X_{t-1}, \ldots$.

We shall use (22) to obtain the answer.

Using the standard formulas (14), we have that

$$g_{g} (z) = (1 - \delta z^{-1}) c(z) c(z^{-1})$$
$$g_{x} (z) = c(z) c(z^{-1})$$

Then (22) becomes

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

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

Then

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

and, alternatively,

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

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

Applying formula (25) of the proposition to evaluating (23) with $g(z) = c(z)$ and $h(z^{-1}) = 1 - \delta z^{-1}$ gives

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

or

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

Thus, we have

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

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

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

(27)

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

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

and

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

The problem is to maximize (27) with respect to a contingency plan expressing $y_t$ as a function of information known at $t$, which is assumed to be $(y_{t-1}, y_{t-2}, ..., a_t, a_{t-1}, ...)$. 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

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

(28)

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 (26) and (28), we have the following solution

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

Blaschke factors

The following is a useful piece of mathematics underlying “root flipping”.

Let $\pi(z) = \sum_{j=0}^{m} \pi_j z^j$ and let $z_1, ..., z_k$ be the zeros of $\pi(z)$ that are inside the unit circle, $k < m$. 19
Then define
\[ \theta(z) = \pi(z) \left( \frac{(z_1 z - 1)}{(z - z_1)} \right) \left( \frac{(z_2 z - 1)}{(z - z_2)} \right) \cdots \left( \frac{(z_k z - 1)}{(z - z_k)} \right) \]
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.

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 \( \varepsilon_t \) 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
\[ \mathbb{E}\hat{X}_{t+1} | X_t, X_{t-1}, \ldots = \sum_{j=0}^{\infty} A_{1j}X_{t-j} \]
Find a formula for the \( A_{2j} \)'s in
\[ \mathbb{E}\hat{X}_{t+2} | X_t, X_{t-1}, \ldots = \sum_{j=0}^{\infty} A_{2j}X_{t-j} \]

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 \( \mathbb{E}U_t = 0 \) 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 \( \varepsilon_t \) 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

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

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 $\eta_t$ is an $(n \times 1)$ vector white noise that is “fundamental” for $X_t$.

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

The optimum predictor of $X_{t+j}$ is

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

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

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

References


