

Classical Control with Linear Algebra

Jesse Perla, Thomas J. Sargent and John Stachurski

December 4, 2020

1 Contents

- Overview [2](#)
- A Control Problem [3](#)
- Finite Horizon Theory [4](#)
- The Infinite Horizon Limit [5](#)
- Undiscounted Problems [6](#)
- Implementation [7](#)
- Exercises [8](#)

2 Overview

In an earlier lecture [Linear Quadratic Dynamic Programming Problems](#) we have studied how to solve a special class of dynamic optimization and prediction problems by applying the method of dynamic programming. In this class of problems

- the objective function is **quadratic** in **states** and **controls**
- the one-step transition function is **linear**
- shocks are i.i.d. Gaussian or martingale differences

In this lecture and a companion lecture [Classical Filtering with Linear Algebra](#), we study the classical theory of linear-quadratic (LQ) optimal control problems.

The classical approach does not use the two closely related methods – dynamic programming and Kalman filtering – that we describe in other lectures, namely, [Linear Quadratic Dynamic Programming Problems](#) and [A First Look at the Kalman Filter](#).

Instead they use either.

- z -transform and lag operator methods, or.
- matrix decompositions applied to linear systems of first-order conditions for optimum problems.

In this lecture and the sequel [Classical Filtering with Linear Algebra](#), we mostly rely on elementary linear algebra.

The main tool from linear algebra we'll put to work here is [LU decomposition](#).

We'll begin with discrete horizon problems.

Then we'll view infinite horizon problems as appropriate limits of these finite horizon problems.

Later, we will examine the close connection between LQ control and least squares prediction and filtering problems.

These classes of problems are connected in the sense that to solve each, essentially the same mathematics is used.

2.1 References

Useful references include [6], [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 Polynomials, Plots, Random, Parameters
        using LinearAlgebra, Statistics
```

3 A Control Problem

Let L be the **lag operator**, so that, for sequence $\{x_t\}$ we have $Lx_t = x_{t-1}$.

More generally, let $L^k x_t = x_{t-k}$ with $L^0 x_t = x_t$ and

$$d(L) = d_0 + d_1 L + \dots + d_m L^m$$

where d_0, d_1, \dots, d_m is a given scalar sequence.

Consider the discrete time control problem

$$\max_{\{y_t\}} \lim_{N \rightarrow \infty} \sum_{t=0}^N \beta^t \left\{ a_t y_t - \frac{1}{2} h y_t^2 - \frac{1}{2} [d(L)y_t]^2 \right\}, \quad (1)$$

where

- h is a positive parameter and $\beta \in (0, 1)$ is a discount factor
- $\{a_t\}_{t \geq 0}$ is a sequence of exponential order less than $\beta^{-1/2}$, by which we mean $\lim_{t \rightarrow \infty} \beta^{t/2} a_t = 0$

Maximization in (1) is subject to initial conditions for $y_{-1}, y_{-2}, \dots, y_{-m}$.

Maximization is over infinite sequences $\{y_t\}_{t \geq 0}$.

3.1 Example

The formulation of the LQ problem given above is broad enough to encompass many useful models.

As a simple illustration, recall that in `lqcontrol` we consider a monopolist facing stochastic demand shocks and adjustment costs.

Let's consider a deterministic version of this problem, where the monopolist maximizes the discounted sum

$$\sum_{t=0}^{\infty} \beta^t \pi_t$$

and

$$\pi_t = p_t q_t - c q_t - \gamma (q_{t+1} - q_t)^2 \quad \text{with} \quad p_t = \alpha_0 - \alpha_1 q_t + d_t$$

In this expression, q_t is output, c is average cost of production, and d_t is a demand shock.

The term $\gamma (q_{t+1} - q_t)^2$ represents adjustment costs.

You will be able to confirm that the objective function can be rewritten as (1) when

- $a_t := \alpha_0 + d_t - c$
- $h := 2\alpha_1$
- $d(L) := \sqrt{2\gamma}(I - L)$

Further examples of this problem for factor demand, economic growth, and government policy problems are given in ch. IX of [5].

4 Finite Horizon Theory

We first study a finite N version of the problem.

Later we will study an infinite horizon problem solution as a limiting version of a finite horizon problem.

(This will require being careful because the limits as $N \rightarrow \infty$ of the necessary and sufficient conditions for maximizing finite N versions of (1) are not sufficient for maximizing (1))

We begin by

1. fixing $N > m$,
2. differentiating the finite version of (1) with respect to y_0, y_1, \dots, y_N , and
3. setting these derivatives to zero

For $t = 0, \dots, N - m$ these first-order necessary conditions are the *Euler equations*.

For $t = N - m + 1, \dots, N$, the first-order conditions are a set of *terminal conditions*.

Consider the term

That is, for the finite N problem, conditions (4) and (5) are necessary and sufficient for a maximum, by concavity of the objective function.

Next we describe how to obtain the solution using matrix methods.

4.1 Matrix Methods

Let's look at how linear algebra can be used to tackle and shed light on the finite horizon LQ control problem.

4.1.1 A Single Lag Term

Let's begin with the special case in which $m = 1$.

We want to solve the system of $N + 1$ linear equations

$$\begin{aligned} [h + d(\beta L^{-1})d(L)]y_t &= a_t, \quad t = 0, 1, \dots, N-1 \\ \beta^N [a_N - h y_N - d_0 d(L)y_N] &= 0 \end{aligned} \quad (6)$$

where $d(L) = d_0 + d_1 L$.

These equations are to be solved for y_0, y_1, \dots, y_N as functions of a_0, a_1, \dots, a_N and y_{-1} .

Let

$$\phi(L) = \phi_0 + \phi_1 L + \beta \phi_1 L^{-1} = h + d(\beta L^{-1})d(L) = (h + d_0^2 + d_1^2) + d_1 d_0 L + d_1 d_0 \beta L^{-1}$$

Then we can represent (6) as the matrix equation

$$\begin{bmatrix} (\phi_0 - d_1^2) & \phi_1 & 0 & 0 & \dots & \dots & 0 \\ \beta \phi_1 & \phi_0 & \phi_1 & 0 & \dots & \dots & 0 \\ 0 & \beta \phi_1 & \phi_0 & \phi_1 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & \beta \phi_1 & \phi_0 & \phi_1 \\ 0 & \dots & \dots & \dots & 0 & \beta \phi_1 & \phi_0 \end{bmatrix} \begin{bmatrix} y_N \\ y_{N-1} \\ y_{N-2} \\ \vdots \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} a_N \\ a_{N-1} \\ a_{N-2} \\ \vdots \\ a_1 \\ a_0 - \phi_1 y_{-1} \end{bmatrix} \quad (7)$$

or

$$W \bar{y} = \bar{a} \quad (8)$$

Notice how we have chosen to arrange the y_t 's in reverse time order.

The matrix W on the left side of (7) is "almost" a [Toeplitz matrix](#) (where each descending diagonal is constant).

There are two sources of deviation from the form of a Toeplitz matrix.

1. The first element differs from the remaining diagonal elements, reflecting the terminal condition.
2. The subdiagonal elements equal β time the superdiagonal elements.

The solution of (8) can be expressed in the form

$$\bar{y} = W^{-1}\bar{a} \quad (9)$$

which represents each element y_t of \bar{y} as a function of the entire vector \bar{a} .

That is, y_t is a function of past, present, and future values of a 's, as well as of the initial condition y_{-1} .

4.1.2 An Alternative Representation

An alternative way to express the solution to (7) or (8) is in so called **feedback-feedforward** form.

The idea here is to find a solution expressing y_t as a function of *past* y 's and *current* and *future* a 's.

To achieve this solution, one can use an **LU decomposition** of W .

There always exists a decomposition of W of the form $W = LU$ where

- L is an $(N + 1) \times (N + 1)$ lower triangular matrix
- U is an $(N + 1) \times (N + 1)$ upper triangular matrix.

The factorization can be normalized so that the diagonal elements of U are unity.

Using the LU representation in (9), we obtain

$$U\bar{y} = L^{-1}\bar{a} \quad (10)$$

Since L^{-1} is lower triangular, this representation expresses y_t as a function of

- lagged y 's (via the term $U\bar{y}$), and
- current and future a 's (via the term $L^{-1}\bar{a}$)

Because there are zeros everywhere in the matrix on the left of (7) except on the diagonal, superdiagonal, and subdiagonal, the LU decomposition takes

- L to be zero except in the diagonal and the leading subdiagonal
- U to be zero except on the diagonal and the superdiagonal

Thus, (10) has the form

$$\begin{bmatrix} 1 & U_{12} & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & U_{23} & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & U_{34} & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & U_{N,N+1} \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_N \\ y_{N-1} \\ y_{N-2} \\ y_{N-3} \\ \vdots \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} L_{11}^{-1} & 0 & 0 & \dots & 0 \\ L_{21}^{-1} & L_{22}^{-1} & 0 & \dots & 0 \\ L_{31}^{-1} & L_{32}^{-1} & L_{33}^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_{N,1}^{-1} & L_{N,2}^{-1} & L_{N,3}^{-1} & \dots & 0 \\ L_{N+1,1}^{-1} & L_{N+1,2}^{-1} & L_{N+1,3}^{-1} & \dots & L_{N+1,N+1}^{-1} \end{bmatrix} \begin{bmatrix} a_N \\ a_{N-1} \\ a_{N-2} \\ \vdots \\ a_1 \\ a_0 - \phi_1 y_{-1} \end{bmatrix}$$

where L_{ij}^{-1} is the (i, j) element of L^{-1} and U_{ij} is the (i, j) element of U .

Note how the left side for a given t involves y_t and one lagged value y_{t-1} while the right side involves all future values of the forcing process a_t, a_{t+1}, \dots, a_N .

4.1.3 Additional Lag Terms

We briefly indicate how this approach extends to the problem with $m > 1$.

Assume that $\beta = 1$ and let D_{m+1} be the $(m+1) \times (m+1)$ symmetric matrix whose elements are determined from the following formula:

$$D_{jk} = d_0 d_{k-j} + d_1 d_{k-j+1} + \dots + d_{j-1} d_{k-1}, \quad k \geq j$$

Let I_{m+1} be the $(m+1) \times (m+1)$ identity matrix.

Let ϕ_j be the coefficients in the expansion $\phi(L) = h + d(L^{-1})d(L)$.

Then the first order conditions (4) and (5) can be expressed as:

$$(D_{m+1} + hI_{m+1}) \begin{bmatrix} y_N \\ y_{N-1} \\ \vdots \\ y_{N-m} \end{bmatrix} = \begin{bmatrix} a_N \\ a_{N-1} \\ \vdots \\ a_{N-m} \end{bmatrix} + M \begin{bmatrix} y_{N-m+1} \\ y_{N-m-2} \\ \vdots \\ y_{N-2m} \end{bmatrix}$$

where M is $(m+1) \times m$ and

$$M_{ij} = \begin{cases} D_{i-j, m+1} & \text{for } i > j \\ 0 & \text{for } i \leq j \end{cases}$$

$$\begin{aligned} \phi_m y_{N-1} + \phi_{m-1} y_{N-2} + \dots + \phi_0 y_{N-m-1} + \phi_1 y_{N-m-2} + \\ \dots + \phi_m y_{N-2m-1} &= a_{N-m-1} \\ \phi_m y_{N-2} + \phi_{m-1} y_{N-3} + \dots + \phi_0 y_{N-m-2} + \phi_1 y_{N-m-3} + \\ \dots + \phi_m y_{N-2m-2} &= a_{N-m-2} \\ &\vdots \\ \phi_m y_{m+1} + \phi_{m-1} y_m + \dots + \phi_0 y_1 + \phi_1 y_0 + \phi_m y_{-m+1} &= a_1 \\ \phi_m y_m + \phi_{m-1} y_{m-1} + \phi_{m-2} + \dots + \phi_0 y_0 + \phi_1 y_{-1} + \dots + \phi_m y_{-m} &= a_0 \end{aligned}$$

As before, we can express this equation as $W\bar{y} = \bar{a}$.

The matrix on the left of this equation is ‘‘almost’’ Toeplitz, the exception being the leading $m \times m$ sub matrix in the upper left hand corner.

We can represent the solution in feedback-feedforward form by obtaining a decomposition $LU = W$, and obtain

$$U\bar{y} = L^{-1}\bar{a} \tag{11}$$

$$\sum_{j=0}^t U_{-t+N+1, -t+N+1-j} y_{t-j} = \sum_{j=0}^{N-t} L_{-t+N+1, -t+N+1-j} \bar{a}_{t+j},$$

$$t = 0, 1, \dots, N$$

where $L_{t,s}^{-1}$ is the element in the (t, s) position of L , and similarly for U .

The left side of equation (11) is the “feedback” part of the optimal control law for y_t , while the right-hand side is the “feedforward” part.

We note that there is a different control law for each t .

Thus, in the finite horizon case, the optimal control law is time dependent.

It is natural to suspect that as $N \rightarrow \infty$, (11) becomes equivalent to the solution of our infinite horizon problem, which below we shall show can be expressed as

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

so that as $N \rightarrow \infty$ we expect that for each fixed t , $U_{t,t-j}^{-1} \rightarrow c_j$ and $L_{t,t+j}$ approaches the coefficient on L^{-j} in the expansion of $c(\beta L^{-1})^{-1}$.

This suspicion is true under general conditions that we shall study later.

For now, we note that by creating the matrix W for large N and factoring it into the LU form, good approximations to $c(L)$ and $c(\beta L^{-1})^{-1}$ can be obtained.

5 The Infinite Horizon Limit

For the infinite horizon problem, we propose to discover first-order necessary conditions by taking the limits of (4) and (5) as $N \rightarrow \infty$.

This approach is valid, and the limits of (4) and (5) as N approaches infinity are first-order necessary conditions for a maximum.

However, for the infinite horizon problem with $\beta < 1$, the limits of (4) and (5) are, in general, not sufficient for a maximum.

That is, the limits of (5) do not provide enough information uniquely to determine the solution of the Euler equation (4) that maximizes (1).

As we shall see below, a side condition on the path of y_t that together with (4) is sufficient for an optimum is

$$\sum_{t=0}^{\infty} \beta^t h y_t^2 < \infty \tag{12}$$

All paths that satisfy the Euler equations, except the one that we shall select below, violate this condition and, therefore, evidently lead to (much) lower values of (1) than does the optimal path selected by the solution procedure below.

Consider the *characteristic equation* associated with the Euler equation

$$h + d(\beta z^{-1}) d(z) = 0 \tag{13}$$

Notice that if \tilde{z} is a root of equation (13), then so is $\beta\tilde{z}^{-1}$.

Thus, the roots of (13) come in “ β -reciprocal” pairs.

Assume that the roots of (13) are distinct.

Let the roots be, in descending order according to their moduli, z_1, z_2, \dots, z_{2m} .

From the reciprocal pairs property and the assumption of distinct roots, it follows that $|z_j| > \sqrt{\beta}$ for $j \leq m$ and $|z_j| < \sqrt{\beta}$ for $j > m$.

It also follows that $z_{2m-j} = \beta z_{j+1}^{-1}, j = 0, 1, \dots, m-1$.

Therefore, the characteristic polynomial on the left side of (13) can be expressed as

$$\begin{aligned} h + d(\beta z^{-1})d(z) &= z^{-m} z_0 (z - z_1) \cdots (z - z_m) (z - z_{m+1}) \cdots (z - z_{2m}) \\ &= z^{-m} z_0 (z - z_1) (z - z_2) \cdots (z - z_m) (z - \beta z_m^{-1}) \cdots (z - \beta z_2^{-1}) (z - \beta z_1^{-1}) \end{aligned} \quad (14)$$

where z_0 is a constant.

In (14), we substitute $(z - z_j) = -z_j(1 - \frac{z}{z_j})$ and $(z - \beta z_j^{-1}) = z(1 - \frac{\beta}{z_j} z^{-1})$ for $j = 1, \dots, m$ to get

$$h + d(\beta z^{-1})d(z) = (-1)^m (z_0 z_1 \cdots z_m) (1 - \frac{z}{z_1}) \cdots (1 - \frac{z}{z_m}) (1 - \frac{\beta}{z_1} z^{-1}) \cdots (1 - \frac{\beta}{z_m} z^{-1})$$

Now define $c(z) = \sum_{j=0}^m c_j z^j$ as

$$c(z) = [(-1)^m z_0 z_1 \cdots z_m]^{1/2} (1 - \frac{z}{z_1}) (1 - \frac{z}{z_2}) \cdots (1 - \frac{z}{z_m}) \quad (15)$$

Notice that (14) can be written

$$h + d(\beta z^{-1})d(z) = c(\beta z^{-1})c(z) \quad (16)$$

It is useful to write (15) as

$$c(z) = c_0 (1 - \lambda_1 z) \cdots (1 - \lambda_m z) \quad (17)$$

where

$$c_0 = [(-1)^m z_0 z_1 \cdots z_m]^{1/2}; \quad \lambda_j = \frac{1}{z_j}, \quad j = 1, \dots, m$$

Since $|z_j| > \sqrt{\beta}$ for $j = 1, \dots, m$ it follows that $|\lambda_j| < 1/\sqrt{\beta}$ for $j = 1, \dots, m$.

Using (17), we can express the factorization (16) as

$$h + d(\beta z^{-1})d(z) = c_0^2 (1 - \lambda_1 z) \cdots (1 - \lambda_m z) (1 - \lambda_1 \beta z^{-1}) \cdots (1 - \lambda_m \beta z^{-1})$$

In sum, we have constructed a factorization (16) of the characteristic polynomial for the Euler equation in which the zeros of $c(z)$ exceed $\beta^{1/2}$ in modulus, and the zeros of $c(\beta z^{-1})$ are less than $\beta^{1/2}$ in modulus.

Using (16), we now write the Euler equation as

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

The unique solution of the Euler equation that satisfies condition (12) is

$$c(L) y_t = c(\beta L^{-1})^{-1} a_t \quad (18)$$

This can be established by using an argument paralleling that in chapter IX of [5].

To exhibit the solution in a form paralleling that of [5], we use (17) to write (18) as

$$(1 - \lambda_1 L) \cdots (1 - \lambda_m L) y_t = \frac{c_0^{-2} a_t}{(1 - \beta \lambda_1 L^{-1}) \cdots (1 - \beta \lambda_m L^{-1})} \quad (19)$$

Using [partial fractions](#), we can write the characteristic polynomial on the right side of (19) as

$$\sum_{j=1}^m \frac{A_j}{1 - \lambda_j \beta L^{-1}} \quad \text{where} \quad A_j := \frac{c_0^{-2}}{\prod_{i \neq j} (1 - \frac{\lambda_i}{\lambda_j})}$$

Then (19) can be written

$$(1 - \lambda_1 L) \cdots (1 - \lambda_m L) y_t = \sum_{j=1}^m \frac{A_j}{1 - \lambda_j \beta L^{-1}} a_t$$

or

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

Equation (20) expresses the optimum sequence for y_t in terms of m lagged y 's, and m weighted infinite geometric sums of future a_t 's.

Furthermore, (20) is the unique solution of the Euler equation that satisfies the initial conditions and condition (12).

In effect, condition (12) compels us to solve the “unstable” roots of $h + d(\beta z^{-1})d(z)$ forward (see [5]).

The step of factoring the polynomial $h + d(\beta z^{-1})d(z)$ into $c(\beta z^{-1})c(z)$, where the zeros of $c(z)$ all have modulus exceeding $\sqrt{\beta}$, is central to solving the problem.

We note two features of the solution (20)

- Since $|\lambda_j| < 1/\sqrt{\beta}$ for all j , it follows that $(\lambda_j \beta) < \sqrt{\beta}$.
- The assumption that $\{a_t\}$ is of exponential order less than $1/\sqrt{\beta}$ is sufficient to guarantee that the geometric sums of future a_t 's on the right side of (20) converge.

We immediately see that those sums will converge under the weaker condition that $\{a_t\}$ is of exponential order less than ϕ^{-1} where $\phi = \max\{\beta \lambda_i, i = 1, \dots, m\}$.

Note that with a_t identically zero, (20) implies that in general $|y_t|$ eventually grows exponentially at a rate given by $\max_i |\lambda_i|$.

The condition $\max_i |\lambda_i| < 1/\sqrt{\beta}$ guarantees that condition (12) is satisfied.

In fact, $\max_i |\lambda_i| < 1/\sqrt{\beta}$ is a necessary condition for (12) to hold.

Were (12) not satisfied, the objective function would diverge to $-\infty$, implying that the y_t path could not be optimal.

For example, with $a_t = 0$, for all t , it is easy to describe a naive (nonoptimal) policy for $\{y_t, t \geq 0\}$ that gives a finite value of (17).

We can simply let $y_t = 0$ for $t \geq 0$.

This policy involves at most m nonzero values of hy_t^2 and $[d(L)y_t]^2$, and so yields a finite value of (1).

Therefore it is easy to dominate a path that violates (12).

6 Undiscounted Problems

It is worthwhile focusing on a special case of the LQ problems above: the undiscounted problem that emerges when $\beta = 1$.

In this case, the Euler equation is

$$\left(h + d(L^{-1})d(L)\right) y_t = a_t$$

The factorization of the characteristic polynomial (16) becomes

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

where

$$\begin{aligned} c(z) &= c_0(1 - \lambda_1 z) \dots (1 - \lambda_m z) \\ c_0 &= \left[(-1)^m z_0 z_1 \dots z_m\right] \\ |\lambda_j| &< 1 \text{ for } j = 1, \dots, m \\ \lambda_j &= \frac{1}{z_j} \text{ for } j = 1, \dots, m \\ z_0 &= \text{constant} \end{aligned}$$

The solution of the problem becomes

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

6.1 Transforming discounted to undiscounted problem

Discounted problems can always be converted into undiscounted problems via a simple transformation.

Consider problem (1) with $0 < \beta < 1$.

Define the transformed variables

$$\tilde{a}_t = \beta^{t/2} a_t, \quad \tilde{y}_t = \beta^{t/2} y_t \quad (21)$$

Then notice that $\beta^t [d(L)y_t]^2 = [\tilde{d}(L)\tilde{y}_t]^2$ with $\tilde{d}(L) = \sum_{j=0}^m \tilde{d}_j L^j$ and $\tilde{d}_j = \beta^{j/2} d_j$.

Then the original criterion function (1) is equivalent to

$$\lim_{N \rightarrow \infty} \sum_{t=0}^N \left\{ \tilde{a}_t \tilde{y}_t - \frac{1}{2} h \tilde{y}_t^2 - \frac{1}{2} [\tilde{d}(L) \tilde{y}_t]^2 \right\} \quad (22)$$

which is to be maximized over sequences $\{\tilde{y}_t, t = 0, \dots\}$ subject to $\tilde{y}_{-1}, \dots, \tilde{y}_{-m}$ given and $\{\tilde{a}_t, t = 1, \dots\}$ a known bounded sequence.

The Euler equation for this problem is $[h + \tilde{d}(L^{-1})\tilde{d}(L)]\tilde{y}_t = \tilde{a}_t$.

The solution is

$$(1 - \tilde{\lambda}_1 L) \cdots (1 - \tilde{\lambda}_m L) \tilde{y}_t = \sum_{j=1}^m \tilde{A}_j \sum_{k=0}^{\infty} \tilde{\lambda}_j^k \tilde{a}_{t+k}$$

or

$$\tilde{y}_t = \tilde{f}_1 \tilde{y}_{t-1} + \cdots + \tilde{f}_m \tilde{y}_{t-m} + \sum_{j=1}^m \tilde{A}_j \sum_{k=0}^{\infty} \tilde{\lambda}_j^k \tilde{a}_{t+k}, \quad (23)$$

where $\tilde{c}(z^{-1})\tilde{c}(z) = h + \tilde{d}(z^{-1})\tilde{d}(z)$, and where

$$[(-1)^m \tilde{z}_0 \tilde{z}_1 \cdots \tilde{z}_m]^{1/2} (1 - \tilde{\lambda}_1 z) \cdots (1 - \tilde{\lambda}_m z) = \tilde{c}(z), \quad \text{where } |\tilde{\lambda}_j| < 1$$

We leave it to the reader to show that (23) implies the equivalent form of the solution

$$y_t = f_1 y_{t-1} + \cdots + f_m y_{t-m} + \sum_{j=1}^m A_j \sum_{k=0}^{\infty} (\lambda_j \beta)^k a_{t+k}$$

where

$$f_j = \tilde{f}_j \beta^{-j/2}, \quad A_j = \tilde{A}_j, \quad \lambda_j = \tilde{\lambda}_j \beta^{-1/2} \quad (24)$$

The transformations (21) and the inverse formulas (24) allow us to solve a discounted problem by first solving a related undiscounted problem.

7 Implementation

Code that computes solutions to the LQ problem using the methods described above can be found in file [control_and_filter.jl](#).

Here's how it looks

```

In [3]: 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  $\phi$  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  $\phi_r$ 
    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 h_eps != nothing
            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)

    @unpack d, m = lqf
    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) Construct 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)
@unpack  $\phi$ , h = lqf

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)] =  $\phi$ '
end

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

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

return W, W_m
end

function roots_of_characteristic(lqf)
    @unpack m,  $\phi$  = lqf

    # Calculate the roots of the 2m-polynomial
     $\phi$ _poly=Polynomial( $\phi$ [end:-1:1])
    proots = roots( $\phi$ _poly)

    # sort the roots according to their length (in descending order)
    roots_sorted = sort(proots, by=abs)[end:-1:1]
    z_0 = sum( $\phi$ ) / (fromroots(proots))(1.0)
    z_1_to_m = roots_sorted[1:m] # we need only those outside the unit circle

     $\lambda$  = 1 ./ z_1_to_m
    return z_1_to_m, z_0,  $\lambda$ 
end

function coeffs_of_c(lqf)
    m = lqf.m
    z_1_to_m, z_0,  $\lambda$  = roots_of_characteristic(lqf)
    c_0 = (z_0 * prod(z_1_to_m) * (-1.0)^m)^(0.5)
    c_coeffs = coeffs(Polynomial(z_1_to_m)) * z_0 / c_0
end

```

```

    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)
    @unpack φ_r, k = lqf
    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(lqf, 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(lqf, N + 1)

    aux_matrix = zeros(N + 1, N + 1)
    aux_matrix[1:t+1, 1:t+1] .= I + zeros(t+1, t+1)
    L = chol(V)'
    Ea_hist = inv(L) * aux_matrix * L * a_hist

    return Ea_hist
end

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

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

    F = lu(W)

    L, U = F.L, F.U
    D = Diagonal(1.0./diag(U))
    U = D * U
    L = L * Diagonal(1.0./diag(D))

```

```

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

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

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

    a_bar = a_hist - W_m * y_m # a_bar from the lecture
    Uy = \ (L, a_bar) # U @ y_bar = L^{-1} a_bar from the lecture
    y_bar = \ (U, Uy) # y_bar = U^{-1} L^{-1} a_bar

    # Reverse the order of y_bar with the matrix J
    J = reverse(I + zeros(N+m+1, N + m + 1), dims = 2)
    y_hist = J * vcat(y_bar, y_m) # y_hist : concatenated y_m and y_bar
    # transform the optimal sequence back if beta is given
    if beta != 1
        y_hist = y_hist .* beta.^(- collect(-m:N)/2)
    end

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

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

    # Reverse the order of y_bar with the matrix J
    J = reverse(I + zeros(N + m + 1, N + m + 1), dims = 2)
    y_hist = J * vcat(y_bar, y_m) # y_hist : concatenated y_m and y_bar
end
return y_hist, L, U, y_bar
end

```

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

7.1 Example

In this application we'll have one lag, with

$$d(L)y_t = \gamma(I - L)y_t = \gamma(y_t - y_{t-1})$$

Suppose for the moment that $\gamma = 0$.

Then the intertemporal component of the LQ problem disappears, and the agent simply wants to maximize $a_t y_t - h y_t^2 / 2$ in each period.

This means that the agent chooses $y_t = a_t / h$.

In the following we'll set $h = 1$, so that the agent just wants to track the $\{a_t\}$ process.

However, as we increase γ , the agent gives greater weight to a smooth time path.

Hence $\{y_t\}$ evolves as a smoothed version of $\{a_t\}$.

The $\{a_t\}$ sequence we'll choose as a stationary cyclic process plus some white noise.

Here's some code that generates a plot when $\gamma = 0.8$

```
In [4]: gr(fmt=:png);

# set seed and generate a_t sequence
Random.seed!(123)
n = 100
a_seq = sin.(range(0, 5 * pi, length = n)) .+ 2 + 0.1 * randn(n)

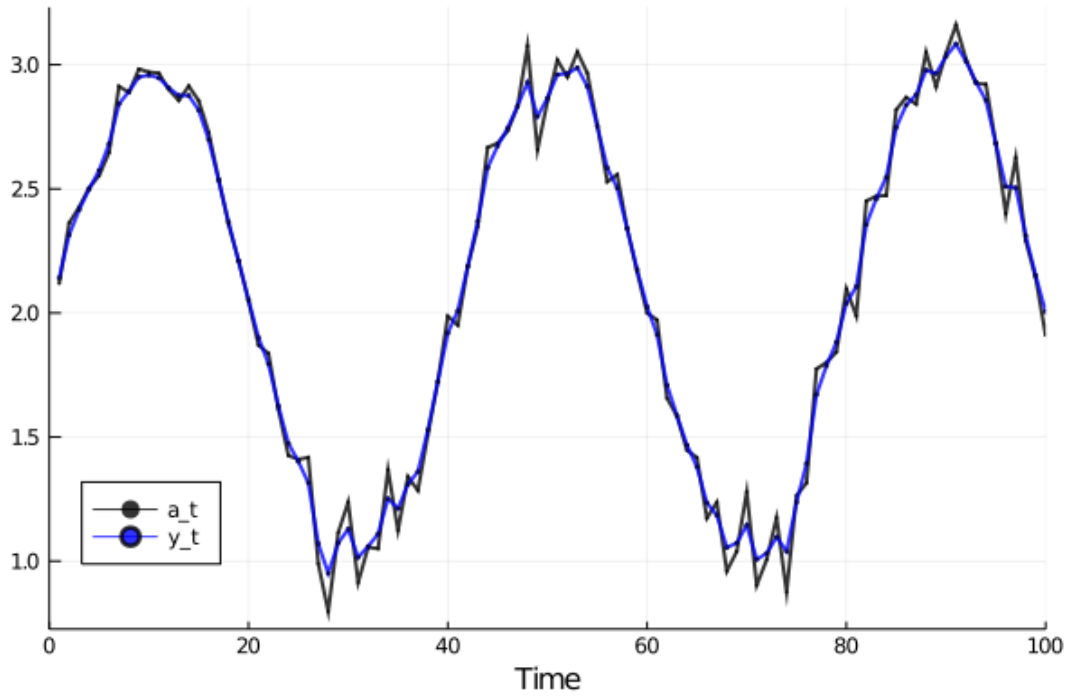
function plot_simulation(;gamma=0.8, m=1, h=1., y_m=2.)
    d = gamma * [1, -1]
    y_m = [y_m]

    testlq = LQFilter(d, h, y_m)
    y_hist, L, U, y = optimal_y(testlq, a_seq)
    y = y[end:-1:1] # reverse y

    # plot simulation results
    time = 1:length(y)
    plt = plot(time, a_seq / h, lw=2, color=:black, alpha=0.8, marker = :
↪circle,
                markersize = 2, label="a_t")
    plot!(plt, time, y, lw=2, color=:blue, marker = :circle, markersize = 2,
↪alpha=0.8,
            label="y_t")
    plot!(plt, xlabel="Time", grid=true, xlim=(0,maximum(time)), legend=:
↪bottomleft)
    end

    plot_simulation()
```

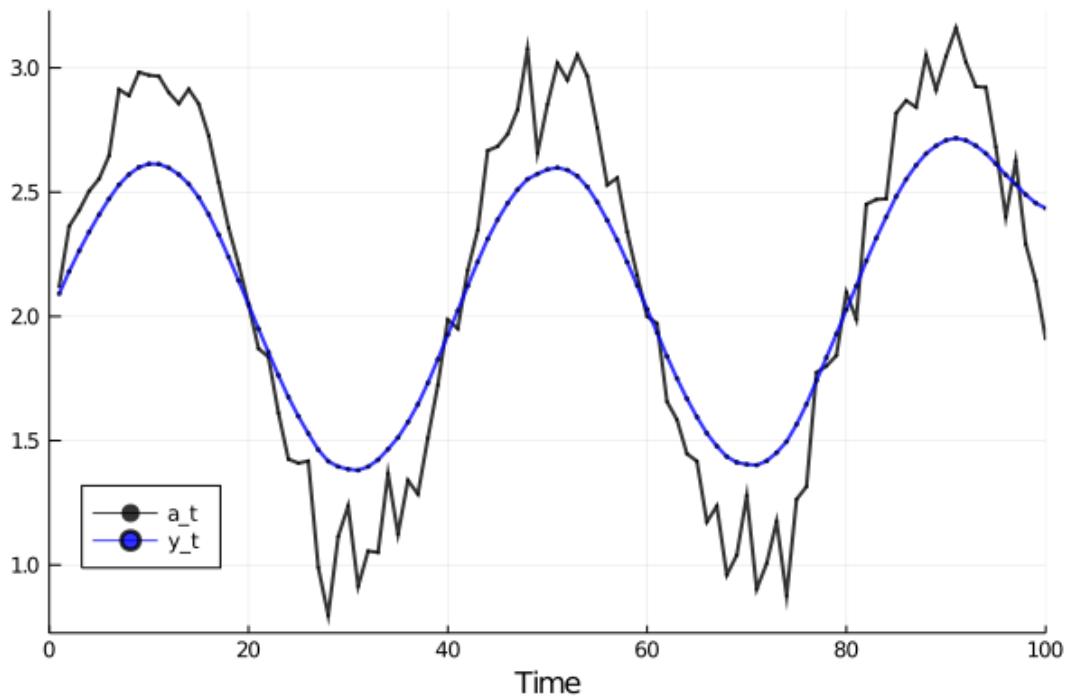
Out[4]:



Here's what happens when we change γ to 5.0

In [5]: `plot_simulation($\gamma=5.0$)`

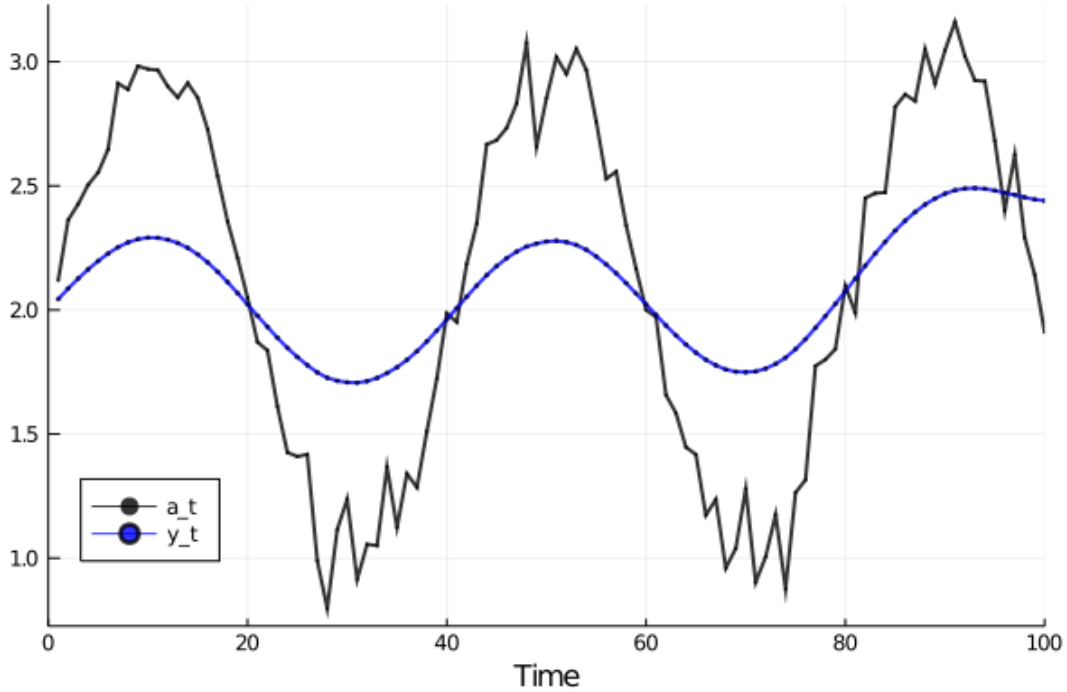
Out[5]:



And here's $\gamma = 10$

In [6]: `plot_simulation($\gamma=10.0$)`

Out[6]:



8 Exercises

8.1 Exercise 1

Consider solving a discounted version ($\beta < 1$) of problem (1), as follows.

Convert (1) to the undiscounted problem (22).

Let the solution of (22) in feedback form be

$$(1 - \tilde{\lambda}_1 L) \cdots (1 - \tilde{\lambda}_m L) \tilde{y}_t = \sum_{j=1}^m \tilde{A}_j \sum_{k=0}^{\infty} \tilde{\lambda}_j^k \tilde{a}_{t+k}$$

or

$$\tilde{y}_t = \tilde{f}_1 \tilde{y}_{t-1} + \cdots + \tilde{f}_m \tilde{y}_{t-m} + \sum_{j=1}^m \tilde{A}_j \sum_{k=0}^{\infty} \tilde{\lambda}_j^k \tilde{a}_{t+k} \quad (25)$$

Here

- $h + \tilde{d}(z^{-1})\tilde{d}(z) = \tilde{c}(z^{-1})\tilde{c}(z)$
- $\tilde{c}(z) = [(-1)^m \tilde{z}_0 \tilde{z}_1 \cdots \tilde{z}_m]^{1/2} (1 - \tilde{\lambda}_1 z) \cdots (1 - \tilde{\lambda}_m z)$

where the \tilde{z}_j are the zeros of $h + \tilde{d}(z^{-1})\tilde{d}(z)$.

Prove that (25) implies that the solution for y_t in feedback form is

$$y_t = f_1 y_{t-1} + \dots + f_m y_{t-m} + \sum_{j=1}^m A_j \sum_{k=0}^{\infty} \beta^k \lambda_j^k a_{t+k}$$

where $f_j = \tilde{f}_j \beta^{-j/2}$, $A_j = \tilde{A}_j$, and $\lambda_j = \tilde{\lambda}_j \beta^{-1/2}$.

8.2 Exercise 2

Solve the optimal control problem, maximize

$$\sum_{t=0}^2 \left\{ a_t y_t - \frac{1}{2} [(1 - 2L)y_t]^2 \right\}$$

subject to y_{-1} given, and $\{a_t\}$ a known bounded sequence.

Express the solution in the “feedback form” (20), giving numerical values for the coefficients.

Make sure that the boundary conditions (5) are satisfied.

(Note: this problem differs from the problem in the text in one important way: instead of $h > 0$ in (1), $h = 0$. This has an important influence on the solution.)

8.3 Exercise 3

Solve the infinite time optimal control problem to maximize

$$\lim_{N \rightarrow \infty} \sum_{t=0}^N -\frac{1}{2} [(1 - 2L)y_t]^2,$$

subject to y_{-1} given. Prove that the solution is

$$y_t = 2y_{t-1} = 2^{t+1}y_{-1} \quad t > 0$$

8.4 Exercise 4

Solve the infinite time problem, to maximize

$$\lim_{N \rightarrow \infty} \sum_{t=0}^N (.0000001) y_t^2 - \frac{1}{2} [(1 - 2L)y_t]^2$$

subject to y_{-1} given. Prove that the solution $y_t = 2y_{t-1}$ violates condition (12), and so is not optimal.

Prove that the optimal solution is approximately $y_t = .5y_{t-1}$.

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] Thomas J Sargent. *Macroeconomic Theory*. Academic Press, New York, 2nd edition, 1987.
- [6] Peter Whittle. *Prediction and regulation by linear least-square methods*. English Univ. Press, 1963.