Dynamic Linear Economies

Thomas J. Sargent and John Stachurski

May 03, 2024

CONTENTS

This website presents a set of lectures on dynamic linear economies and tools needed for this class of economic models.

Part I

Kalman Filter

CHAPTER

ONE

A FIRST LOOK AT THE KALMAN FILTER

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install quantecon
```

1.1 Overview

This lecture provides a simple and intuitive introduction to the Kalman filter, for those who either

- have heard of the Kalman filter but don't know how it works, or
- know the Kalman filter equations, but don't know where they come from

For additional (more advanced) reading on the Kalman filter, see

- [Ljungqvist and Sargent, 2018], section 2.7
- [Anderson and Moore, 2005]

The second reference presents a comprehensive treatment of the Kalman filter.

Required knowledge: Familiarity with matrix manipulations, multivariate normal distributions, covariance matrices, etc.

We'll need the following imports:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
from scipy import linalg
import numpy as np
import matplotlib.cm as cm
```

(continues on next page)

(continued from previous page)

```
from quantecon import Kalman, LinearStateSpace
from scipy.stats import norm
from scipy.integrate import quad
from scipy.linalg import eigvals
```

1.2 The Basic Idea

The Kalman filter has many applications in economics, but for now let's pretend that we are rocket scientists.

A missile has been launched from country Y and our mission is to track it.

Let $x \in \mathbb{R}^2$ denote the current location of the missile—a pair indicating latitude-longitude coordinates on a map.

At the present moment in time, the precise location x is unknown, but we do have some beliefs about x.

One way to summarize our knowledge is a point prediction $\hat{x}$

- But what if the President wants to know the probability that the missile is currently over the Sea of Japan?
- Then it is better to summarize our initial beliefs with a bivariate probability density p
  - $\int_E p(x)dx$ indicates the probability that we attach to the missile being in region E.

The density p is called our *prior* for the random variable x.

To keep things tractable in our example, we assume that our prior is Gaussian.

In particular, we take

$$p = N(\hat{x}, \Sigma) \tag{1.1}$$

where $\hat{x}$ is the mean of the distribution and Σ is a 2×2 covariance matrix. In our simulations, we will suppose that

$$\hat{x} = \begin{pmatrix} 0.2 \\ -0.2 \end{pmatrix}, \qquad \Sigma = \begin{pmatrix} 0.4 & 0.3 \\ 0.3 & 0.45 \end{pmatrix} \tag{1.2}$$

This density $p(x)$ is shown below as a contour map, with the center of the red ellipse being equal to $\hat{x}$.

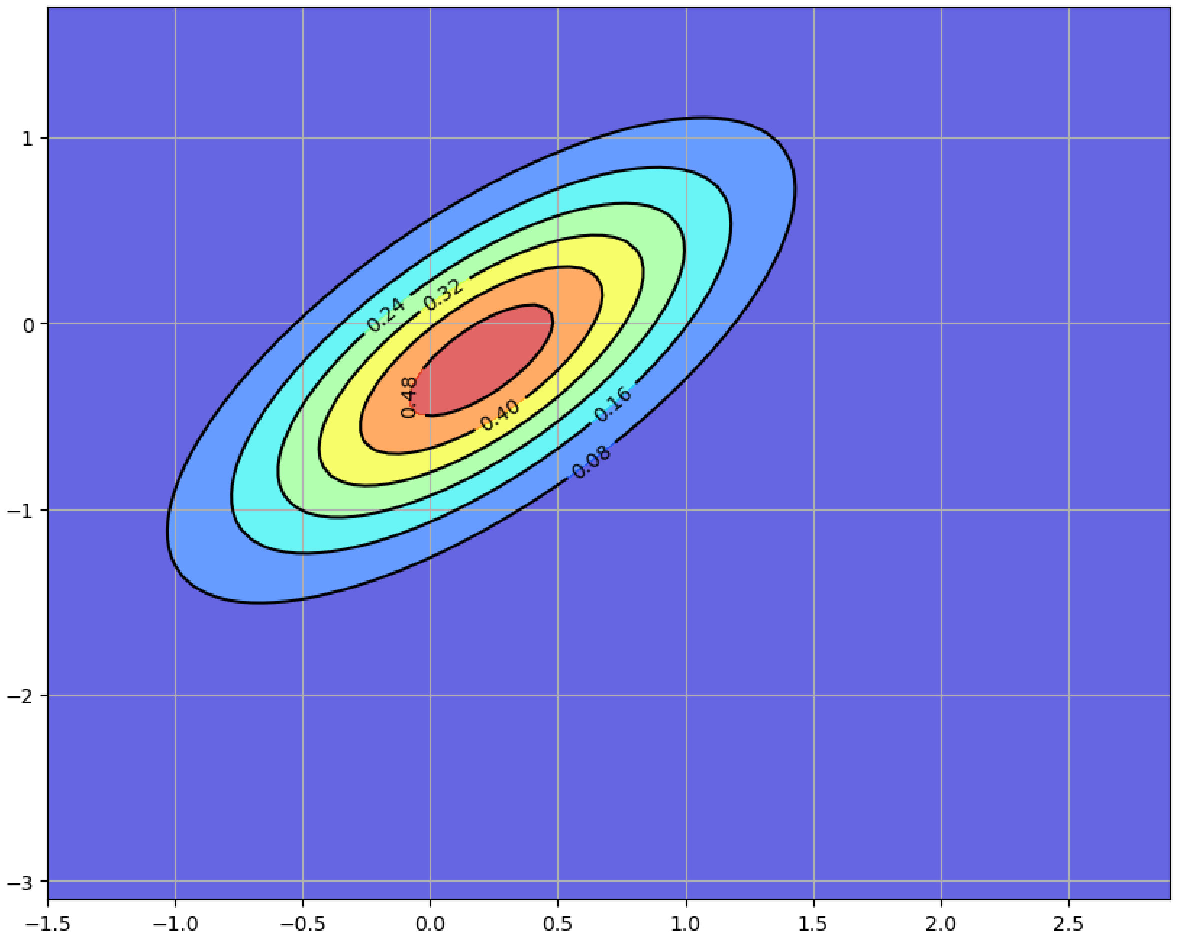
```
# Set up the Gaussian prior density p
Σ = [[0.4, 0.3], [0.3, 0.45]]
Σ = np.matrix(Σ)
x_hat = np.matrix([0.2, -0.2]).T
# Define the matrices G and R from the equation y = G x + N(0, R)
G = [[1, 0], [0, 1]]
G = np.matrix(G)
R = 0.5 * Σ
# The matrices A and Q
A = [[1.2, 0], [0, -0.2]]
A = np.matrix(A)
Q = 0.3 * Σ
# The observed value of y
y = np.matrix([2.3, -1.9]).T

# Set up grid for plotting
x_grid = np.linspace(-1.5, 2.9, 100)
y_grid = np.linspace(-3.1, 1.7, 100)
X, Y = np.meshgrid(x_grid, y_grid)
```

(continues on next page)

(continued from previous page)

```
def bivariate_normal(x, y, σ_x=1.0, σ_y=1.0, μ_x=0.0, μ_y=0.0, σ_xy=0.0):
    """
    Compute and return the probability density function of bivariate normal
    distribution of normal random variables x and y

    Parameters
    ----------
    x : array_like(float)
        Random variable

    y : array_like(float)
        Random variable

    σ_x : array_like(float)
          Standard deviation of random variable x

    σ_y : array_like(float)
          Standard deviation of random variable y

    μ_x : scalar(float)
          Mean value of random variable x

    μ_y : scalar(float)
          Mean value of random variable y

    σ_xy : array_like(float)
           Covariance of random variables x and y

    """

    x_μ = x - μ_x
    y_μ = y - μ_y

    ρ = σ_xy / (σ_x * σ_y)
    z = x_μ**2 / σ_x**2 + y_μ**2 / σ_y**2 - 2 * ρ * x_μ * y_μ / (σ_x * σ_y)
    denom = 2 * np.pi * σ_x * σ_y * np.sqrt(1 - ρ**2)
    return np.exp(-z / (2 * (1 - ρ**2))) / denom

def gen_gaussian_plot_vals(μ, C):
    "Z values for plotting the bivariate Gaussian N(μ, C)"
    m_x, m_y = float(μ[0]), float(μ[1])
    s_x, s_y = np.sqrt(C[0, 0]), np.sqrt(C[1, 1])
    s_xy = C[0, 1]
    return bivariate_normal(X, Y, s_x, s_y, m_x, m_y, s_xy)

# Plot the figure

fig, ax = plt.subplots(figsize=(10, 8))
ax.grid()

Z = gen_gaussian_plot_vals(x_hat, Σ)
ax.contourf(X, Y, Z, 6, alpha=0.6, cmap=cm.jet)
cs = ax.contour(X, Y, Z, 6, colors="black")
ax.clabel(cs, inline=1, fontsize=10)

plt.show()
```

```
/tmp/ipykernel_6131/3508717107.py:61: DeprecationWarning: Conversion of an array
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
↪extract a single element from your array before performing this operation.
↪(Deprecated NumPy 1.25.)
  m_x, m_y = float(μ[0]), float(μ[1])
```



1.2.1 The Filtering Step

We are now presented with some good news and some bad news.

The good news is that the missile has been located by our sensors, which report that the current location is $y = (2.3, -1.9)$.

The next figure shows the original prior $p(x)$ and the new reported location y

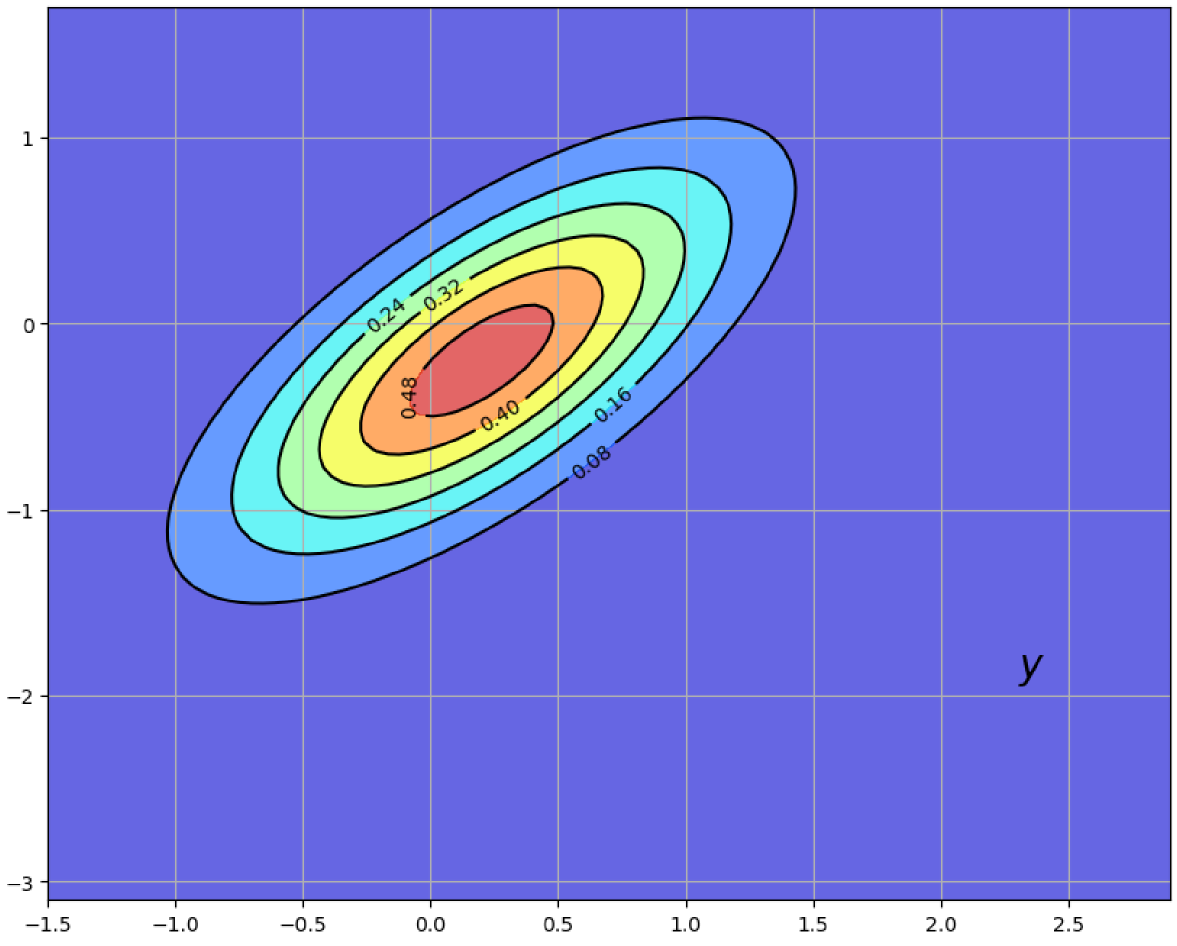
```
fig, ax = plt.subplots(figsize=(10, 8))
ax.grid()

Z = gen_gaussian_plot_vals(x_hat, Σ)
ax.contourf(X, Y, Z, 6, alpha=0.6, cmap=cm.jet)
cs = ax.contour(X, Y, Z, 6, colors="black")
ax.clabel(cs, inline=1, fontsize=10)
ax.text(float(y[0]), float(y[1]), "$y$", fontsize=20, color="black")
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

```
/tmp/ipykernel_6131/3508717107.py:61: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  m_x, m_y = float(μ[0]), float(μ[1])
/tmp/ipykernel_6131/3470248806.py:8: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  ax.text(float(y[0]), float(y[1]), "$y$", fontsize=20, color="black")
```



The bad news is that our sensors are imprecise.

In particular, we should interpret the output of our sensor not as $y = x$, but rather as

$$y = Gx + v, \quad \text{where} \quad v \sim N(0, R) \tag{1.3}$$

Here G and R are 2×2 matrices with R positive definite. Both are assumed known, and the noise term v is assumed to be independent of x.

How then should we combine our prior $p(x) = N(\hat{x}, \Sigma)$ and this new information y to improve our understanding of the location of the missile?

As you may have guessed, the answer is to use Bayes' theorem, which tells us to update our prior $p(x)$ to $p(x \,|\, y)$ via

$$p(x \,|\, y) = \frac{p(y \,|\, x)\, p(x)}{p(y)}$$

where $p(y) = \int p(y \,|\, x)\, p(x) dx$.

In solving for $p(x \,|\, y)$, we observe that

- $p(x) = N(\hat{x}, \Sigma)$.
- In view of (1.3), the conditional density $p(y \,|\, x)$ is $N(Gx, R)$.
- $p(y)$ does not depend on x, and enters into the calculations only as a normalizing constant.

Because we are in a linear and Gaussian framework, the updated density can be computed by calculating population linear regressions.

In particular, the solution is known[1] to be

$$p(x \,|\, y) = N(\hat{x}^F, \Sigma^F)$$

where

$$\hat{x}^F := \hat{x} + \Sigma G' (G \Sigma G' + R)^{-1}(y - G\hat{x}) \quad \text{and} \quad \Sigma^F := \Sigma - \Sigma G' (G \Sigma G' + R)^{-1} G \Sigma \tag{1.4}$$

Here $\Sigma G' (G \Sigma G' + R)^{-1}$ is the matrix of population regression coefficients of the hidden object $x - \hat{x}$ on the surprise $y - G\hat{x}$.

This new density $p(x \,|\, y) = N(\hat{x}^F, \Sigma^F)$ is shown in the next figure via contour lines and the color map.

The original density is left in as contour lines for comparison

```
fig, ax = plt.subplots(figsize=(10, 8))
ax.grid()

Z = gen_gaussian_plot_vals(x_hat, Σ)
cs1 = ax.contour(X, Y, Z, 6, colors="black")
ax.clabel(cs1, inline=1, fontsize=10)
M = Σ * G.T * linalg.inv(G * Σ * G.T + R)
x_hat_F = x_hat + M * (y - G * x_hat)
Σ_F = Σ - M * G * Σ
new_Z = gen_gaussian_plot_vals(x_hat_F, Σ_F)
cs2 = ax.contour(X, Y, new_Z, 6, colors="black")
ax.clabel(cs2, inline=1, fontsize=10)
ax.contourf(X, Y, new_Z, 6, alpha=0.6, cmap=cm.jet)
ax.text(float(y[0]), float(y[1]), "$y$", fontsize=20, color="black")

plt.show()
```

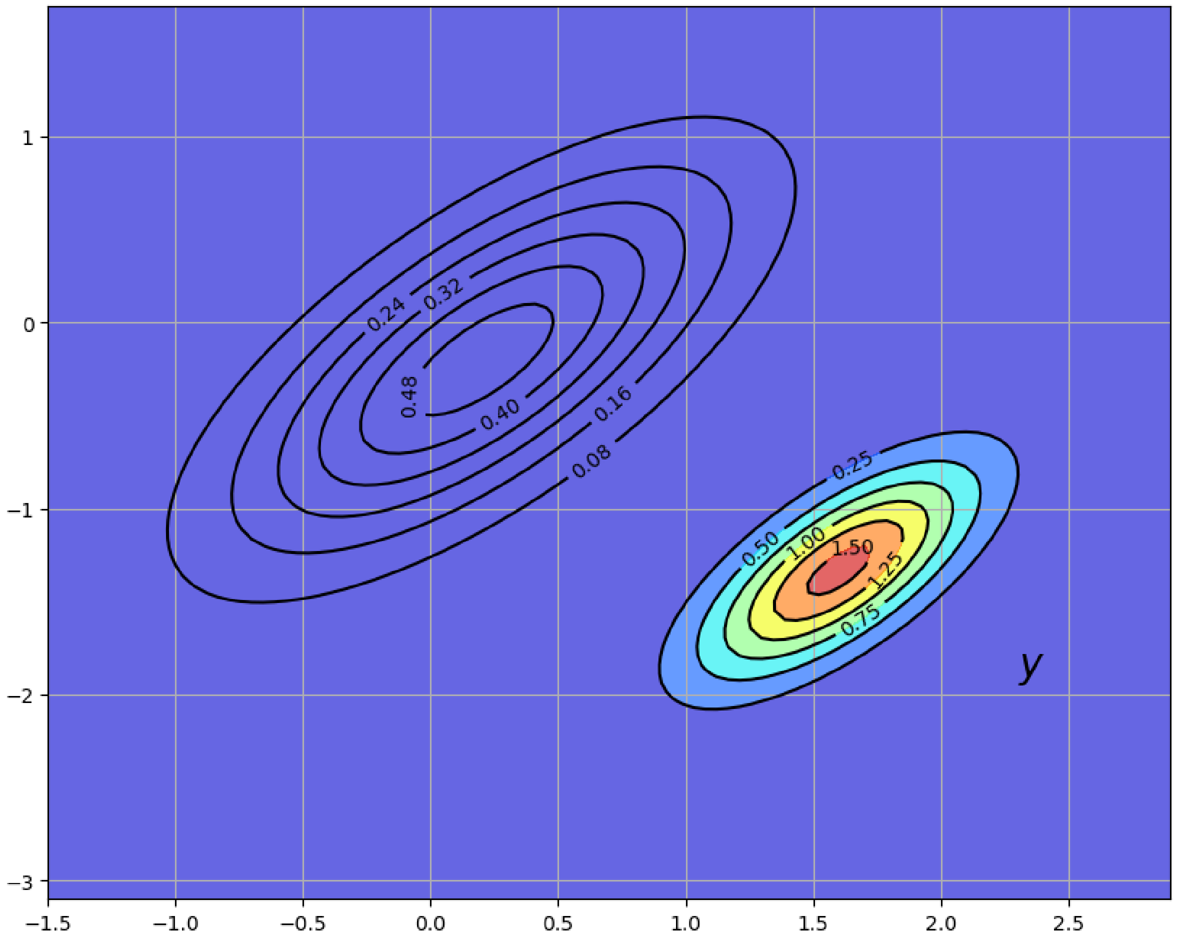
```
/tmp/ipykernel_6131/3508717107.py:61: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  m_x, m_y = float(μ[0]), float(μ[1])
```

(continues on next page)

[1] See, for example, page 93 of [Bishop, 2006]. To get from his expressions to the ones used above, you will also need to apply the Woodbury matrix identity.

(continued from previous page)

```
/tmp/ipykernel_6131/792457825.py:14: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  ax.text(float(y[0]), float(y[1]), "$y$", fontsize=20, color="black")
```



Our new density twists the prior $p(x)$ in a direction determined by the new information $y - G\hat{x}$.

In generating the figure, we set G to the identity matrix and $R = 0.5\Sigma$ for Σ defined in (1.2).

1.2.2 The Forecast Step

What have we achieved so far?

We have obtained probabilities for the current location of the state (missile) given prior and current information.

This is called "filtering" rather than forecasting because we are filtering out noise rather than looking into the future.

- $p(x \mid y) = N(\hat{x}^F, \Sigma^F)$ is called the *filtering distribution*

But now let's suppose that we are given another task: to predict the location of the missile after one unit of time (whatever that may be) has elapsed.

To do this we need a model of how the state evolves.

Let's suppose that we have one, and that it's linear and Gaussian. In particular,

$$x_{t+1} = Ax_t + w_{t+1}, \quad \text{where} \quad w_t \sim N(0, Q) \tag{1.5}$$

Our aim is to combine this law of motion and our current distribution $p(x \mid y) = N(\hat{x}^F, \Sigma^F)$ to come up with a new *predictive* distribution for the location in one unit of time.

In view of (1.5), all we have to do is introduce a random vector $x^F \sim N(\hat{x}^F, \Sigma^F)$ and work out the distribution of $Ax^F + w$ where w is independent of x^F and has distribution $N(0, Q)$.

Since linear combinations of Gaussians are Gaussian, $Ax^F + w$ is Gaussian.

Elementary calculations and the expressions in (1.4) tell us that

$$\mathbb{E}[Ax^F + w] = A\mathbb{E}x^F + \mathbb{E}w = A\hat{x}^F = A\hat{x} + A\Sigma G'(G\Sigma G' + R)^{-1}(y - G\hat{x})$$

and

$$\operatorname{Var}[Ax^F + w] = A \operatorname{Var}[x^F]A' + Q = A\Sigma^F A' + Q = A\Sigma A' - A\Sigma G'(G\Sigma G' + R)^{-1}G\Sigma A' + Q$$

The matrix $A\Sigma G'(G\Sigma G' + R)^{-1}$ is often written as K_Σ and called the *Kalman gain*.

- The subscript Σ has been added to remind us that K_Σ depends on Σ, but not y or $\hat{x}$.

Using this notation, we can summarize our results as follows.

Our updated prediction is the density $N(\hat{x}_{new}, \Sigma_{new})$ where

$$\begin{aligned} \hat{x}_{new} &:= A\hat{x} + K_\Sigma(y - G\hat{x}) \\ \Sigma_{new} &:= A\Sigma A' - K_\Sigma G\Sigma A' + Q \end{aligned}$$

- The density $p_{new}(x) = N(\hat{x}_{new}, \Sigma_{new})$ is called the *predictive distribution*

The predictive distribution is the new density shown in the following figure, where the update has used parameters.

$$A = \begin{pmatrix} 1.2 & 0.0 \\ 0.0 & -0.2 \end{pmatrix}, \qquad Q = 0.3 * \Sigma$$

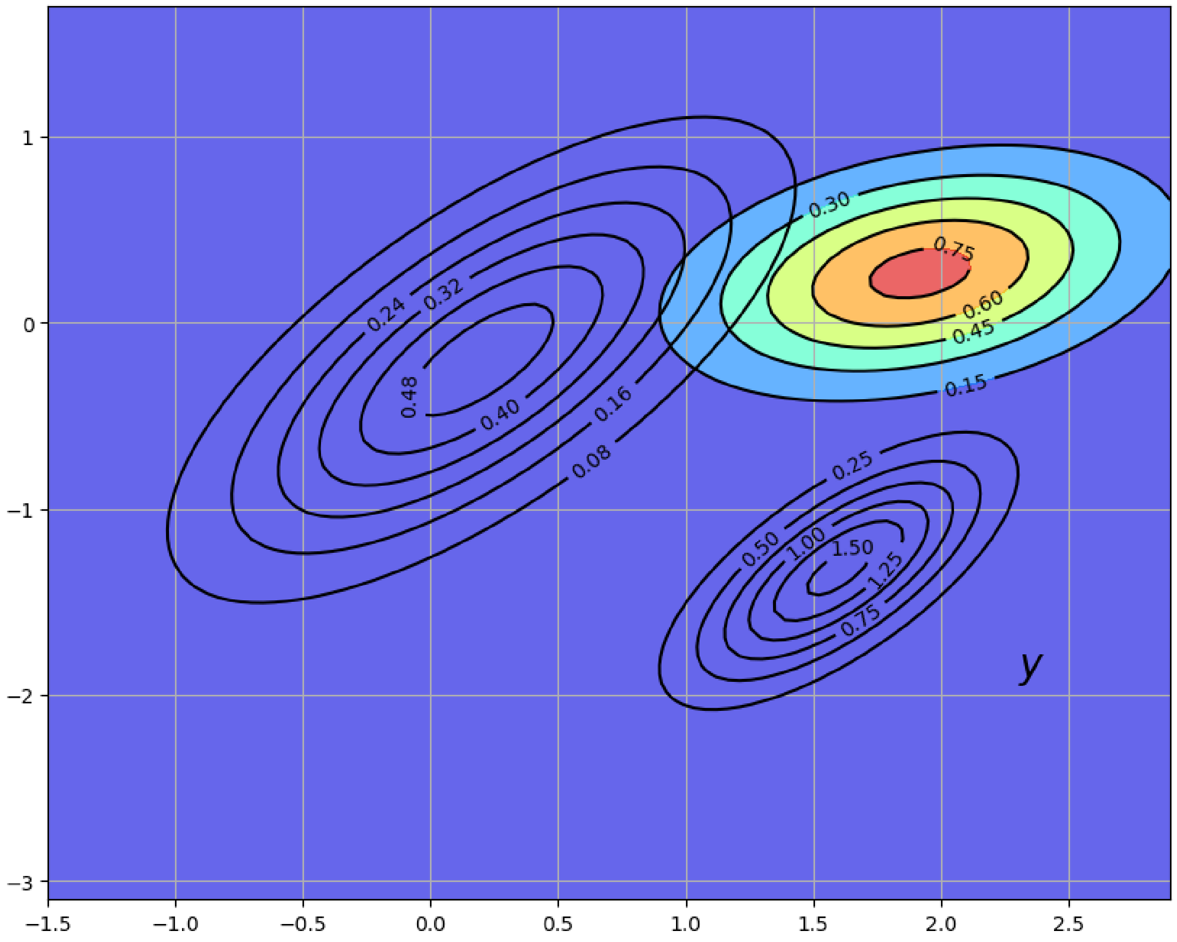
```
fig, ax = plt.subplots(figsize=(10, 8))
ax.grid()

# Density 1
Z = gen_gaussian_plot_vals(x_hat, Σ)
cs1 = ax.contour(X, Y, Z, 6, colors="black")
ax.clabel(cs1, inline=1, fontsize=10)

# Density 2
M = Σ * G.T * linalg.inv(G * Σ * G.T + R)
x_hat_F = x_hat + M * (y - G * x_hat)
Σ_F = Σ - M * G * Σ
Z_F = gen_gaussian_plot_vals(x_hat_F, Σ_F)
cs2 = ax.contour(X, Y, Z_F, 6, colors="black")
ax.clabel(cs2, inline=1, fontsize=10)

# Density 3
new_x_hat = A * x_hat_F
new_Σ = A * Σ_F * A.T + Q
new_Z = gen_gaussian_plot_vals(new_x_hat, new_Σ)
cs3 = ax.contour(X, Y, new_Z, 6, colors="black")
```

(continues on next page)

(continued from previous page)

```
ax.clabel(cs3, inline=1, fontsize=10)
ax.contourf(X, Y, new_Z, 6, alpha=0.6, cmap=cm.jet)
ax.text(float(y[0]), float(y[1]), "$y$", fontsize=20, color="black")

plt.show()
```

```
/tmp/ipykernel_6131/3508717107.py:61: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  m_x, m_y = float(μ[0]), float(μ[1])
/tmp/ipykernel_6131/3056082785.py:24: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  ax.text(float(y[0]), float(y[1]), "$y$", fontsize=20, color="black")
```



1.2.3 The Recursive Procedure

Let's look back at what we've done.

We started the current period with a prior $p(x)$ for the location x of the missile.

We then used the current measurement y to update to $p(x \mid y)$.

Finally, we used the law of motion (1.5) for $\{x_t\}$ to update to $p_{new}(x)$.

If we now step into the next period, we are ready to go round again, taking $p_{new}(x)$ as the current prior.

Swapping notation $p_t(x)$ for $p(x)$ and $p_{t+1}(x)$ for $p_{new}(x)$, the full recursive procedure is:

1. Start the current period with prior $p_t(x) = N(\hat{x}_t, \Sigma_t)$.
2. Observe current measurement y_t.
3. Compute the filtering distribution $p_t(x \mid y) = N(\hat{x}_t^F, \Sigma_t^F)$ from $p_t(x)$ and y_t, applying Bayes rule and the conditional distribution (1.3).
4. Compute the predictive distribution $p_{t+1}(x) = N(\hat{x}_{t+1}, \Sigma_{t+1})$ from the filtering distribution and (1.5).
5. Increment t by one and go to step 1.

Repeating (1.6), the dynamics for $\hat{x}_t$ and Σ_t are as follows

$$\begin{aligned} \hat{x}_{t+1} &= A\hat{x}_t + K_{\Sigma_t}(y_t - G\hat{x}_t) \\ \Sigma_{t+1} &= A\Sigma_t A' - K_{\Sigma_t} G \Sigma_t A' + Q \end{aligned}$$

These are the standard dynamic equations for the Kalman filter (see, for example, [Ljungqvist and Sargent, 2018], page 58).

1.3 Convergence

The matrix Σ_t is a measure of the uncertainty of our prediction $\hat{x}_t$ of x_t.

Apart from special cases, this uncertainty will never be fully resolved, regardless of how much time elapses.

One reason is that our prediction $\hat{x}_t$ is made based on information available at $t-1$, not t.

Even if we know the precise value of x_{t-1} (which we don't), the transition equation (1.5) implies that $x_t = Ax_{t-1} + w_t$.

Since the shock w_t is not observable at $t-1$, any time $t-1$ prediction of x_t will incur some error (unless w_t is degenerate).

However, it is certainly possible that Σ_t converges to a constant matrix as $t \to \infty$.

To study this topic, let's expand the second equation in (1.6):

$$\Sigma_{t+1} = A\Sigma_t A' - A\Sigma_t G'(G\Sigma_t G' + R)^{-1} G\Sigma_t A' + Q \tag{1.6}$$

This is a nonlinear difference equation in Σ_t.

A fixed point of (1.6) is a constant matrix Σ such that

$$\Sigma = A\Sigma A' - A\Sigma G'(G\Sigma G' + R)^{-1} G\Sigma A' + Q \tag{1.7}$$

Equation (1.6) is known as a discrete-time Riccati difference equation.

Equation (1.7) is known as a discrete-time algebraic Riccati equation.

Conditions under which a fixed point exists and the sequence $\{\Sigma_t\}$ converges to it are discussed in [Anderson *et al.*, 1996] and [Anderson and Moore, 2005], chapter 4.

A sufficient (but not necessary) condition is that all the eigenvalues λ_i of A satisfy $|\lambda_i| < 1$ (cf. e.g., [Anderson and Moore, 2005], p. 77).

(This strong condition assures that the unconditional distribution of x_t converges as $t \to +\infty$.)

In this case, for any initial choice of Σ_0 that is both non-negative and symmetric, the sequence $\{\Sigma_t\}$ in (1.6) converges to a non-negative symmetric matrix Σ that solves (1.7).

1.4 Implementation

The class `Kalman` from the QuantEcon.py package implements the Kalman filter

- Instance data consists of:
  - the moments $(\hat{x}_t, \Sigma_t)$ of the current prior.
  - An instance of the LinearStateSpace class from QuantEcon.py.

The latter represents a linear state space model of the form

$$\begin{aligned} x_{t+1} &= Ax_t + Cw_{t+1} \\ y_t &= Gx_t + Hv_t \end{aligned}$$

where the shocks w_t and v_t are IID standard normals.

To connect this with the notation of this lecture we set

$$Q := CC' \quad \text{and} \quad R := HH'$$

- The class `Kalman` from the QuantEcon.py package has a number of methods, some that we will wait to use until we study more advanced applications in subsequent lectures.
- Methods pertinent for this lecture are:
  - `prior_to_filtered`, which updates $(\hat{x}_t, \Sigma_t)$ to $(\hat{x}_t^F, \Sigma_t^F)$
  - `filtered_to_forecast`, which updates the filtering distribution to the predictive distribution – which becomes the new prior $(\hat{x}_{t+1}, \Sigma_{t+1})$
  - `update`, which combines the last two methods
  - a `stationary_values`, which computes the solution to (1.7) and the corresponding (stationary) Kalman gain

You can view the program on GitHub.

1.5 Exercises

Exercise 1.5.1

Consider the following simple application of the Kalman filter, loosely based on [Ljungqvist and Sargent, 2018], section 2.9.2.

Suppose that

- all variables are scalars
- the hidden state $\{x_t\}$ is in fact constant, equal to some $\theta \in \mathbb{R}$ unknown to the modeler

State dynamics are therefore given by (1.5) with $A = 1$, $Q = 0$ and $x_0 = \theta$.

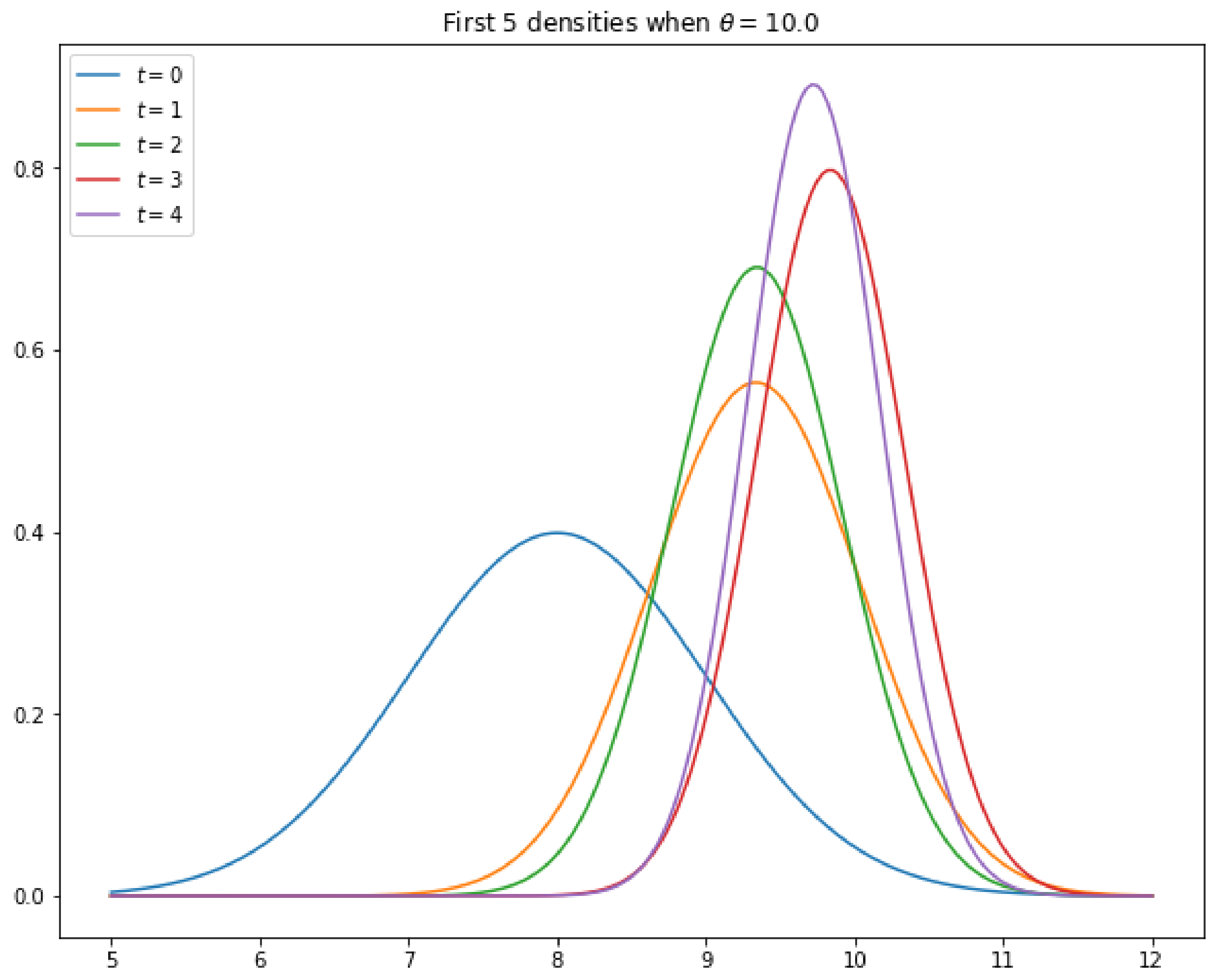
The measurement equation is $y_t = \theta + v_t$ where v_t is $N(0, 1)$ and IID.

The task of this exercise to simulate the model and, using the code from `kalman.py`, plot the first five predictive densities $p_t(x) = N(\hat{x}_t, \Sigma_t)$.

As shown in [Ljungqvist and Sargent, 2018], sections 2.9.1–2.9.2, these distributions asymptotically put all mass on the unknown value θ.

In the simulation, take $\theta = 10$, $\hat{x}_0 = 8$ and $\Sigma_0 = 1$.

Your figure should – modulo randomness – look something like this



Solution to Exercise 1.5.1

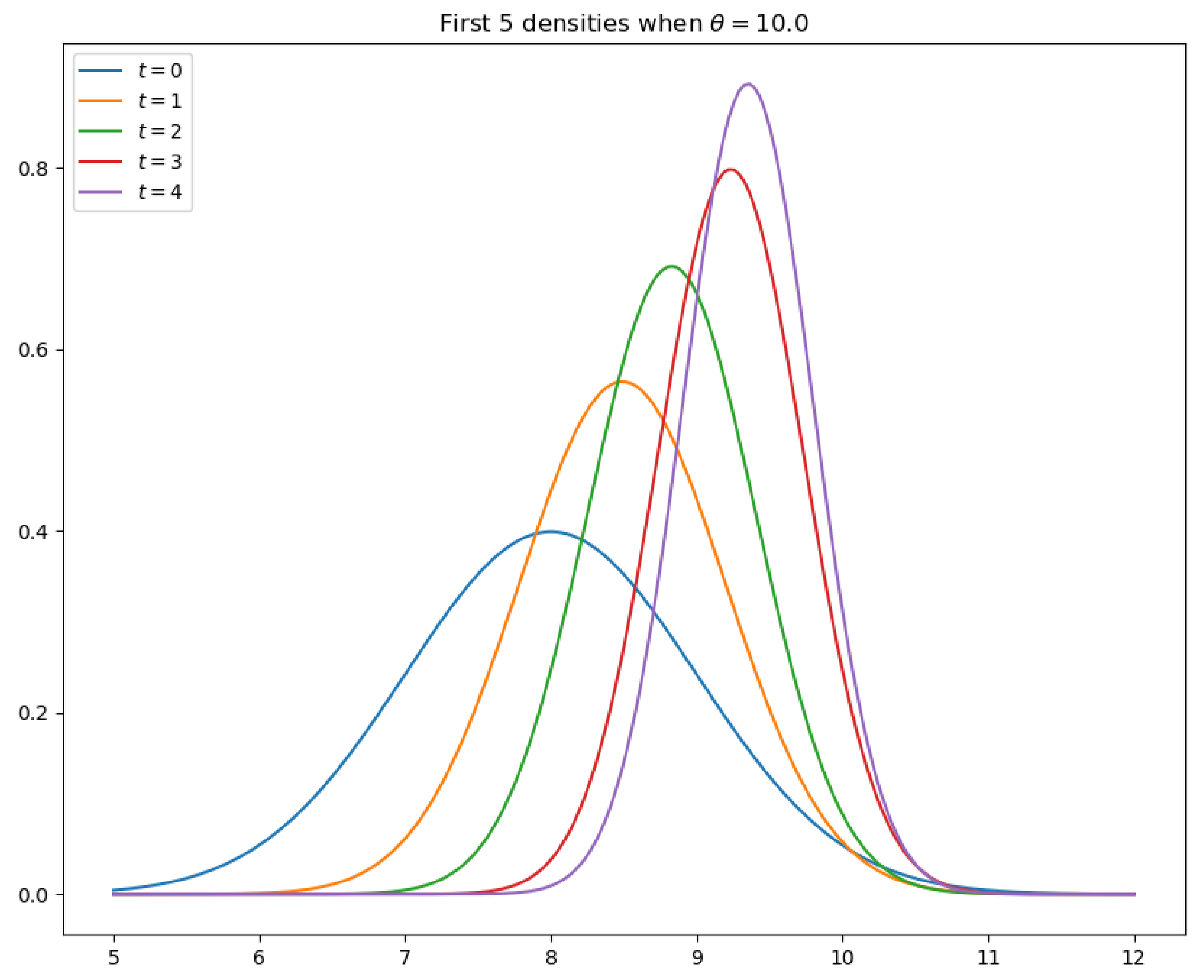
```
# Parameters
θ = 10  # Constant value of state x_t
A, C, G, H = 1, 0, 1, 1
ss = LinearStateSpace(A, C, G, H, mu_0=θ)
```

(continues on next page)

(continued from previous page)

```
# Set prior, initialize kalman filter
x_hat_0, Σ_0 = 8, 1
kalman = Kalman(ss, x_hat_0, Σ_0)

# Draw observations of y from state space model
N = 5
x, y = ss.simulate(N)
y = y.flatten()

# Set up plot
fig, ax = plt.subplots(figsize=(10,8))
xgrid = np.linspace(θ - 5, θ + 2, 200)

for i in range(N):
    # Record the current predicted mean and variance
    m, v = [float(z) for z in (kalman.x_hat, kalman.Sigma)]
    # Plot, update filter
    ax.plot(xgrid, norm.pdf(xgrid, loc=m, scale=np.sqrt(v)), label=f'$t={i}$')
    kalman.update(y[i])

ax.set_title(f'First {N} densities when $\\theta = {θ:.1f}$')
ax.legend(loc='upper left')
plt.show()
```

```
/tmp/ipykernel_6131/1660567565.py:21: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  m, v = [float(z) for z in (kalman.x_hat, kalman.Sigma)]
```



Exercise 1.5.2

The preceding figure gives some support to the idea that probability mass converges to θ.

To get a better idea, choose a small $\epsilon > 0$ and calculate

$$z_t := 1 - \int_{\theta-\epsilon}^{\theta+\epsilon} p_t(x) dx$$

for $t = 0, 1, 2, \ldots, T$.

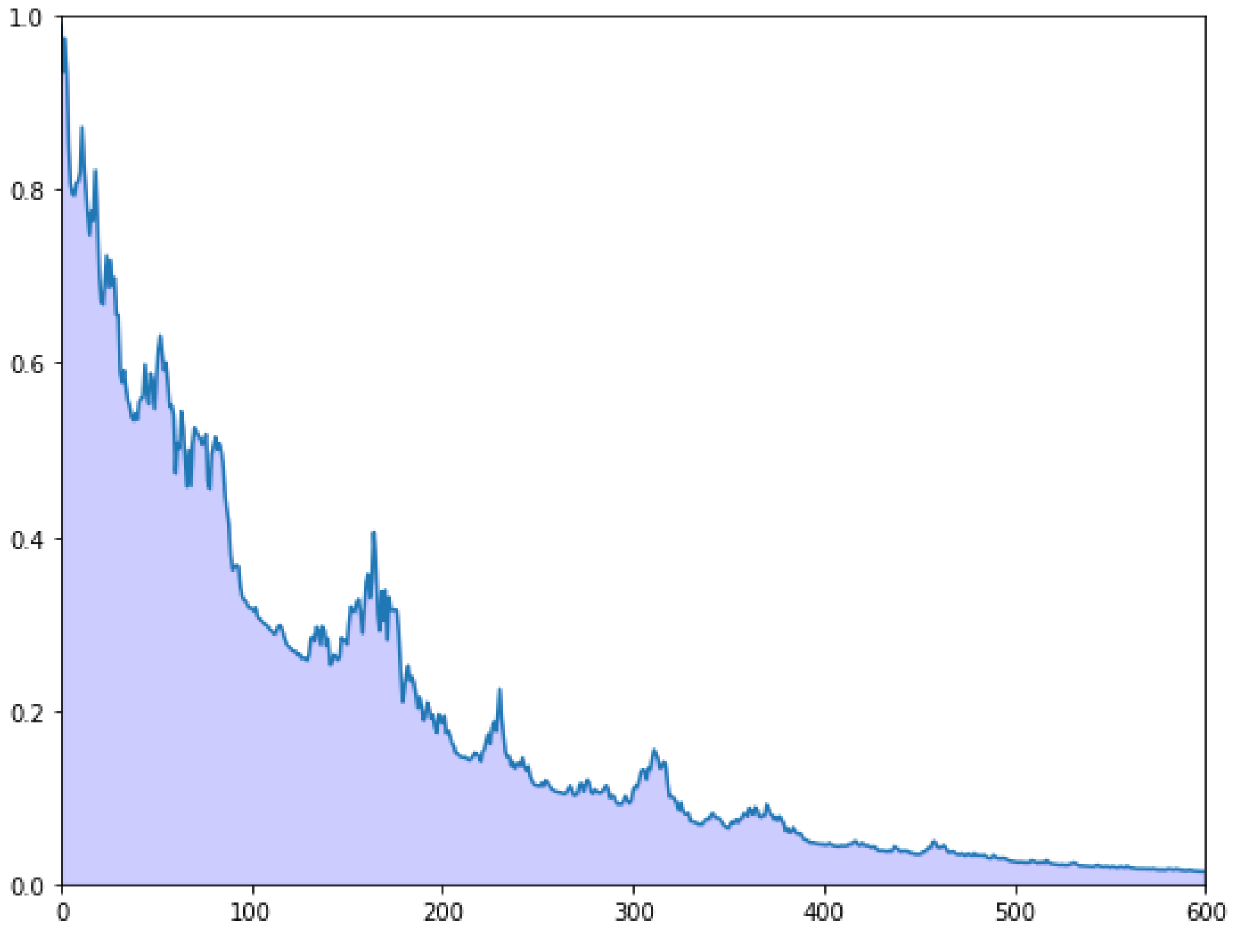
Plot z_t against T, setting $\epsilon = 0.1$ and $T = 600$.

Your figure should show error erratically declining something like this

Solution to Exercise 1.5.2

```
ϵ = 0.1
θ = 10  # Constant value of state x_t
A, C, G, H = 1, 0, 1, 1
ss = LinearStateSpace(A, C, G, H, mu_0=θ)
```

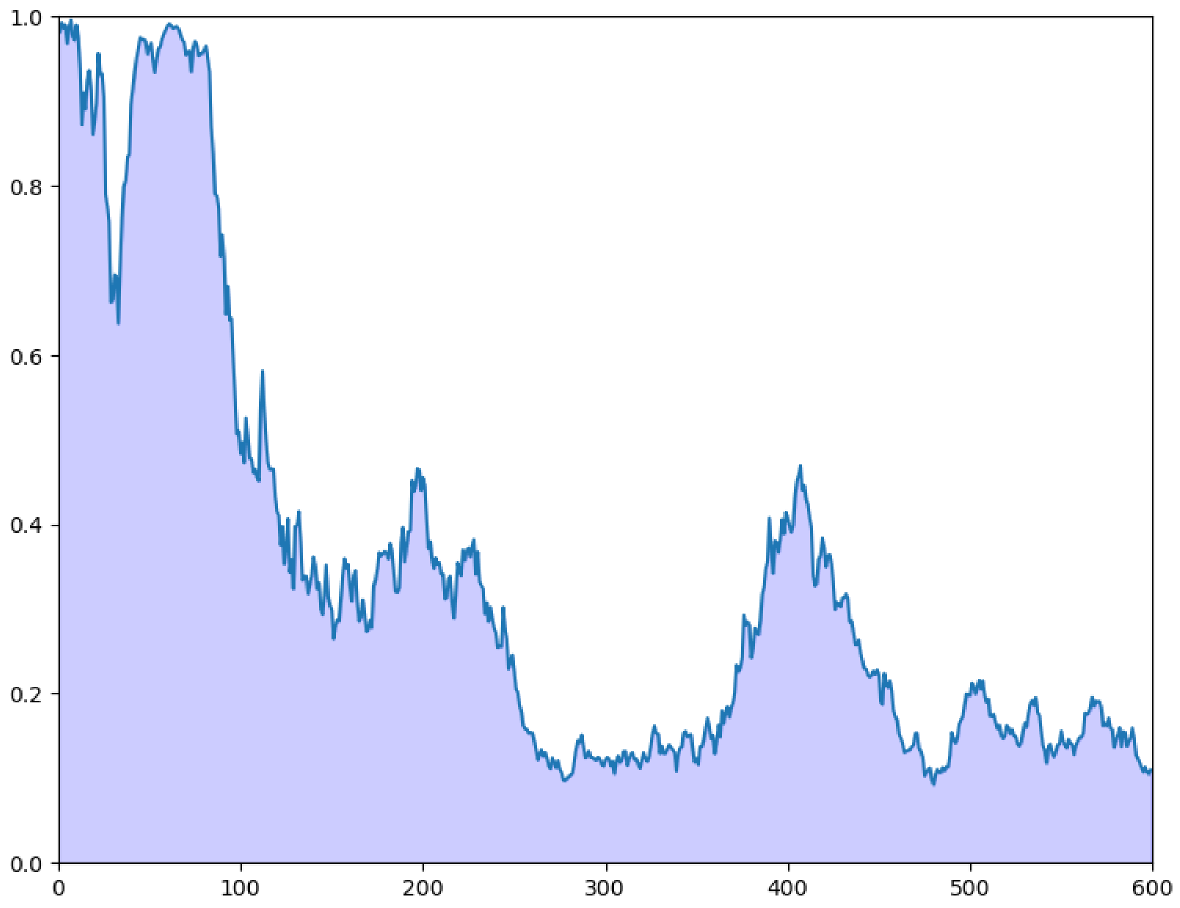
(continues on next page)


1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600

(continued from previous page)

```
x_hat_0, Σ_0 = 8, 1
kalman = Kalman(ss, x_hat_0, Σ_0)

T = 600
z = np.empty(T)
x, y = ss.simulate(T)
y = y.flatten()

for t in range(T):
    # Record the current predicted mean and variance and plot their densities
    m, v = [float(temp) for temp in (kalman.x_hat, kalman.Sigma)]

    f = lambda x: norm.pdf(x, loc=m, scale=np.sqrt(v))
    integral, error = quad(f, θ - ϵ, θ + ϵ)
    z[t] = 1 - integral

    kalman.update(y[t])

fig, ax = plt.subplots(figsize=(9, 7))
ax.set_ylim(0, 1)
ax.set_xlim(0, T)
ax.plot(range(T), z)
ax.fill_between(range(T), np.zeros(T), z, color="blue", alpha=0.2)
plt.show()
```

```
/tmp/ipykernel_6131/3050251196.py:16: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  m, v = [float(temp) for temp in (kalman.x_hat, kalman.Sigma)]
```



Exercise 1.5.3

As discussed *above*, if the shock sequence $\{w_t\}$ is not degenerate, then it is not in general possible to predict x_t without error at time $t-1$ (and this would be the case even if we could observe x_{t-1}).

Let's now compare the prediction $\hat{x}_t$ made by the Kalman filter against a competitor who **is** allowed to observe x_{t-1}.

This competitor will use the conditional expectation $\mathbb{E}[x_t \,|\, x_{t-1}]$, which in this case is Ax_{t-1}.

The conditional expectation is known to be the optimal prediction method in terms of minimizing mean squared error.

(More precisely, the minimizer of $\mathbb{E}\,\|x_t - g(x_{t-1})\|^2$ with respect to g is $g^*(x_{t-1}) := \mathbb{E}[x_t \,|\, x_{t-1}]$)

Thus we are comparing the Kalman filter against a competitor who has more information (in the sense of being able to observe the latent state) and behaves optimally in terms of minimizing squared error.

Our horse race will be assessed in terms of squared error.

In particular, your task is to generate a graph plotting observations of both $\|x_t - Ax_{t-1}\|^2$ and $\|x_t - \hat{x}_t\|^2$ against t for $t = 1, \ldots, 50$.

For the parameters, set $G = I, R = 0.5I$ and $Q = 0.3I$, where I is the 2×2 identity.

Set

$$A = \begin{pmatrix} 0.5 & 0.4 \\ 0.6 & 0.3 \end{pmatrix}$$

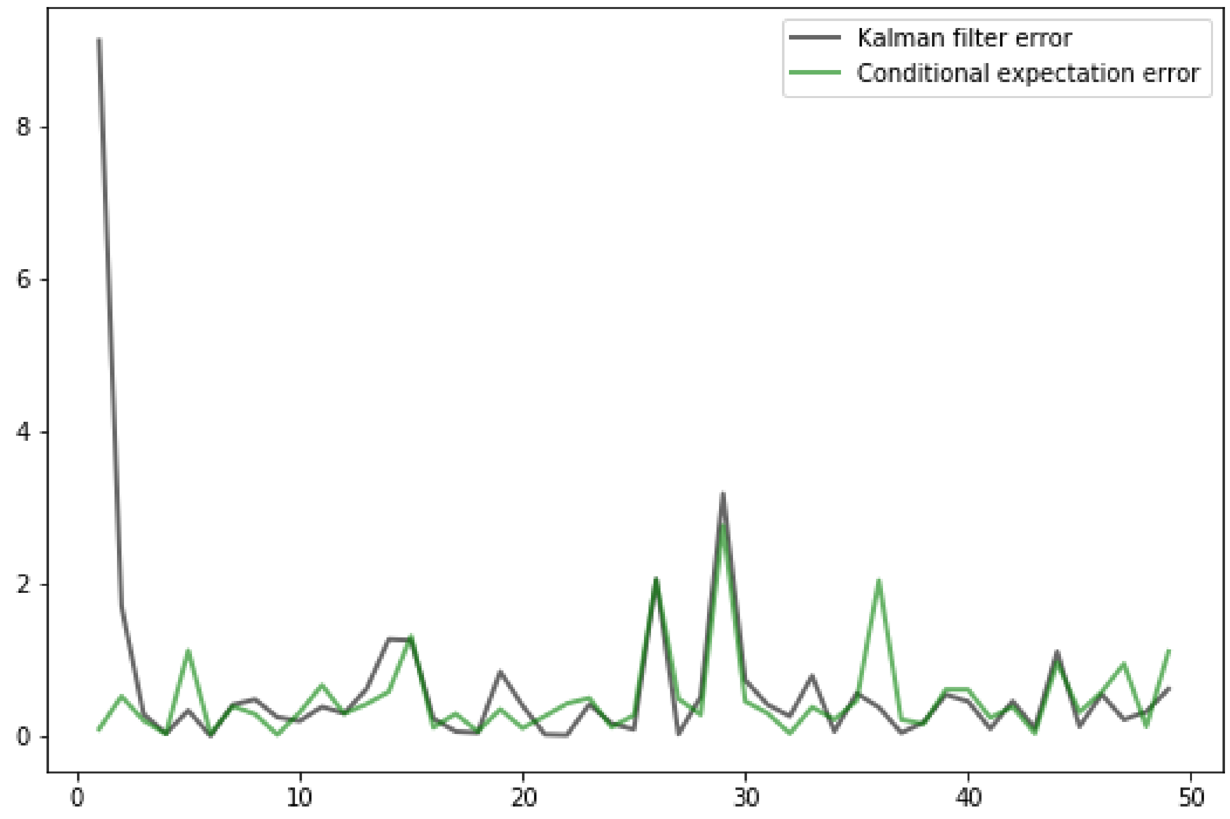
To initialize the prior density, set

$$\Sigma_0 = \begin{pmatrix} 0.9 & 0.3 \\ 0.3 & 0.9 \end{pmatrix}$$

and $\hat{x}_0 = (8, 8)$.

Finally, set $x_0 = (0, 0)$.

You should end up with a figure similar to the following (modulo randomness)



Observe how, after an initial learning period, the Kalman filter performs quite well, even relative to the competitor who predicts optimally with knowledge of the latent state.

Solution to Exercise 1.5.3

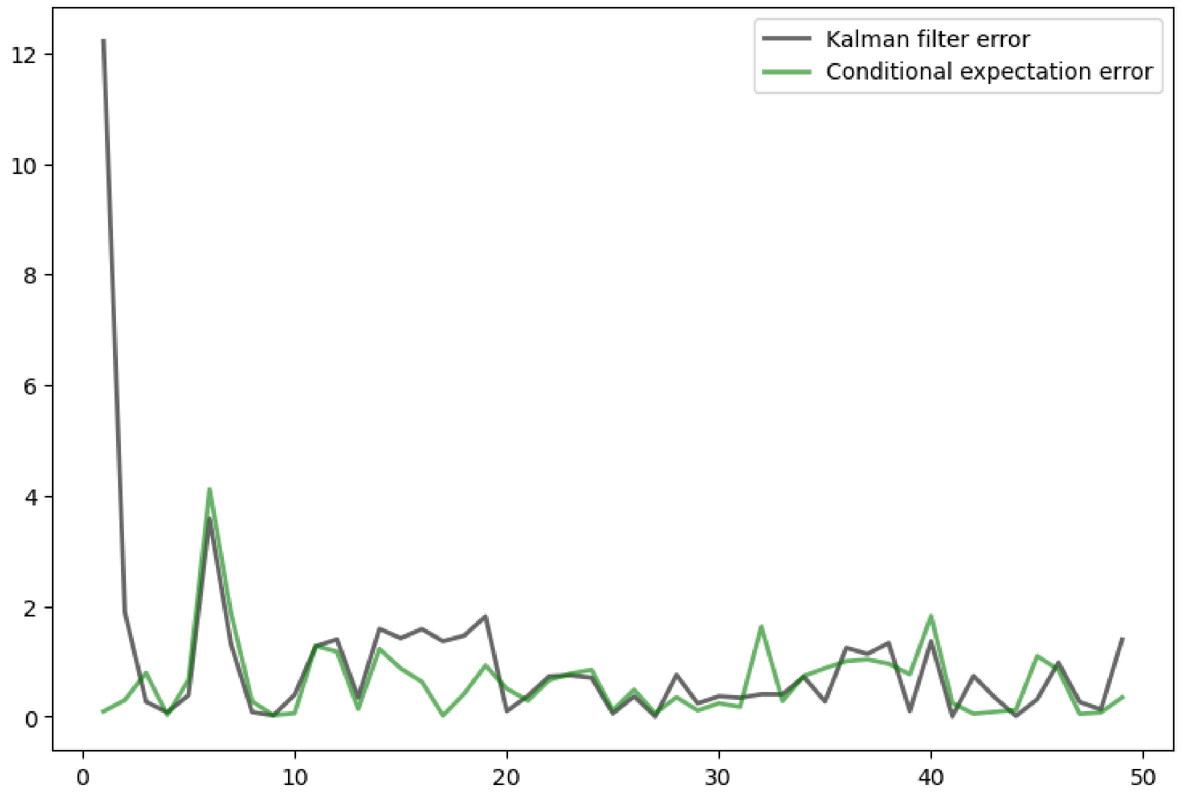
```
# Define A, C, G, H
G = np.identity(2)
H = np.sqrt(0.5) * np.identity(2)

A = [[0.5, 0.4],
     [0.6, 0.3]]
C = np.sqrt(0.3) * np.identity(2)

# Set up state space mode, initial value x_0 set to zero
ss = LinearStateSpace(A, C, G, H, mu_0 = np.zeros(2))
```

(continues on next page)

(continued from previous page)

```python
# Define the prior density
Σ = [[0.9, 0.3],
     [0.3, 0.9]]
Σ = np.array(Σ)
x_hat = np.array([8, 8])

# Initialize the Kalman filter
kn = Kalman(ss, x_hat, Σ)

# Print eigenvalues of A
print("Eigenvalues of A:")
print(eigvals(A))

# Print stationary Σ
S, K = kn.stationary_values()
print("Stationary prediction error variance:")
print(S)

# Generate the plot
T = 50
x, y = ss.simulate(T)

e1 = np.empty(T-1)
e2 = np.empty(T-1)

for t in range(1, T):
    kn.update(y[:,t])
    e1[t-1] = np.sum((x[:, t] - kn.x_hat.flatten())**2)
    e2[t-1] = np.sum((x[:, t] - A @ x[:, t-1])**2)

fig, ax = plt.subplots(figsize=(9,6))
ax.plot(range(1, T), e1, 'k-', lw=2, alpha=0.6,
        label='Kalman filter error')
ax.plot(range(1, T), e2, 'g-', lw=2, alpha=0.6,
        label='Conditional expectation error')
ax.legend()
plt.show()
```

```
Eigenvalues of A:
[ 0.9+0.j -0.1+0.j]
Stationary prediction error variance:
[[0.40329108 0.1050718 ]
 [0.1050718  0.41061709]]
```



Exercise 1.5.4

Try varying the coefficient 0.3 in $Q = 0.3I$ up and down.

Observe how the diagonal values in the stationary solution Σ (see (1.7)) increase and decrease in line with this coefficient.

The interpretation is that more randomness in the law of motion for x_t causes more (permanent) uncertainty in prediction.

CHAPTER

TWO

ANOTHER LOOK AT THE KALMAN FILTER

Contents

In this quantecon lecture *A First Look at the Kalman filter*, we used a Kalman filter to estimate locations of a rocket.

In this lecture, we'll use the Kalman filter to infer a worker's human capital and the effort that the worker devotes to accumulating human capital, neither of which the firm observes directly.

The firm learns about those things only by observing a history of the output that the worker generates for the firm, and from understanding how that output depends on the worker's human capital and how human capital evolves as a function of the worker's effort.

We'll posit a rule that expresses how the much firm pays the worker each period as a function of the firm's information each period.

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install quantecon
```

To conduct simulations, we bring in these imports, as in *A First Look at the Kalman filter*.

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
from quantecon import Kalman, LinearStateSpace
from collections import namedtuple
from scipy.stats import multivariate_normal
import matplotlib as mpl

mpl.rcParams['text.usetex'] = True
mpl.rcParams['text.latex.preamble'] = r'\usepackage{{amsmath}}'
```

2.1 A worker's output

A representative worker is permanently employed at a firm.

The workers' output is described by the following dynamic process:

$$
\begin{aligned}
h_{t+1} &= \alpha h_t + \beta u_t + c w_{t+1}, \quad c_{t+1} \sim \mathcal{N}(0,1) \\
u_{t+1} &= u_t \\
y_t &= g h_t + v_t, \quad v_t \sim \mathcal{N}(0,R)
\end{aligned}
\tag{2.1}
$$

Here

- h_t is the logarithm of human capital at time t
- u_t is the logarithm of the worker's effort at accumulating human capital at t
- y_t is the logarithm of the worker's output at time t
- $h_0 \sim \mathcal{N}(\hat{h}_0, \sigma_{h,0})$
- $u_0 \sim \mathcal{N}(\hat{u}_0, \sigma_{u,0})$

Parameters of the model are $\alpha, \beta, c, R, g, \hat{h}_0, \hat{u}_0, \sigma_h, \sigma_u$.

At time 0, a firm has hired the worker.

The worker is permanently attached to the firm and so works for the same firm at all dates $t = 0, 1, 2, \ldots$.

At the beginning of time 0, the firm observes neither the worker's innate initial human capital h_0 nor its hard-wired permanent effort level u_0.

The firm believes that u_0 for a particular worker is drawn from a Gaussian probability distribution, and so is described by $u_0 \sim \mathcal{N}(\hat{u}_0, \sigma_{u,0})$.

The h_t part of a worker's "type" moves over time, but the effort component of the worker's type is $u_t = u_0$.

This means that from the firm's point of view, the worker's effort is effectively an unknown fixed "parameter".

At time $t \geq 1$, for a particular worker the firm observed $y^{t-1} = [y_{t-1}, y_{t-2}, \ldots, y_0]$.

The firm does not observe the worker's "type" (h_0, u_0).

But the firm does observe the worker's output y_t at time t and remembers the worker's past outputs y^{t-1}.

2.2 A firm's wage-setting policy

Based on information about the worker that the firm has at time $t \geq 1$, the firm pays the worker log wage

$$
w_t = g E[h_t | y^{t-1}], \quad t \geq 1
$$

and at time 0 pays the worker a log wage equal to the unconditional mean of y_0:

$$
w_0 = g \hat{h}_0
$$

In using this payment rule, the firm is taking into account that the worker's log output today is partly due to the random component v_t that comes entirely from luck, and that is assumed to be independent of h_t and u_t.

2.3 A state-space representation

Write system *(2.1.1)* in the state-space form

$$\begin{bmatrix} h_{t+1} \\ u_{t+1} \end{bmatrix} = \begin{bmatrix} \alpha & \beta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} h_t \\ u_t \end{bmatrix} + \begin{bmatrix} c \\ 0 \end{bmatrix} w_{t+1}$$
$$y_t = \begin{bmatrix} g & 0 \end{bmatrix} \begin{bmatrix} h_t \\ u_t \end{bmatrix} + v_t$$

which is equivalent with

$$\begin{aligned} x_{t+1} &= Ax_t + Cw_{t+1} \\ y_t &= Gx_t + v_t \\ x_0 &\sim \mathcal{N}(\hat{x}_0, \Sigma_0) \end{aligned} \tag{2.2}$$

where

$$x_t = \begin{bmatrix} h_t \\ u_t \end{bmatrix}, \quad \hat{x}_0 = \begin{bmatrix} \hat{h}_0 \\ \hat{u}_0 \end{bmatrix}, \quad \Sigma_0 = \begin{bmatrix} \sigma_{h,0} & 0 \\ 0 & \sigma_{u,0} \end{bmatrix}$$

To compute the firm's wage setting policy, we first we create a `namedtuple` to store the parameters of the model

```
WorkerModel = namedtuple("WorkerModel",
                ('A', 'C', 'G', 'R', 'xhat_0', 'Σ_0'))

def create_worker(α=.8, β=.2, c=.2,
                  R=.5, g=1.0, hhat_0=4, uhat_0=4,
                  σ_h=4, σ_u=4):

    A = np.array([[α, β],
                  [0, 1]])
    C = np.array([[c],
                  [0]])
    G = np.array([g, 1])

    # Define initial state and covariance matrix
    xhat_0 = np.array([[hhat_0],
                       [uhat_0]])

    Σ_0 = np.array([[σ_h, 0],
                    [0, σ_u]])

    return WorkerModel(A=A, C=C, G=G, R=R, xhat_0=xhat_0, Σ_0=Σ_0)
```

Please note how the `WorkerModel` namedtuple creates all of the objects required to compute an associated state-space representation (2.2).

This is handy, because in order to simulate a history $\{y_t, h_t\}$ for a worker, we'll want to form state space system for him/her by using the `LinearStateSpace` class.

```
# Define A, C, G, R, xhat_0, Σ_0
worker = create_worker()
A, C, G, R = worker.A, worker.C, worker.G, worker.R
xhat_0, Σ_0 = worker.xhat_0, worker.Σ_0

# Create a LinearStateSpace object
```

(continues on next page)

(continued from previous page)

```python
ss = LinearStateSpace(A, C, G, np.sqrt(R),
        mu_0=xhat_0, Sigma_0=np.zeros((2,2)))

T = 100
x, y = ss.simulate(T)
y = y.flatten()

h_0, u_0 = x[0, 0], x[1, 0]
```

Next, to compute the firm's policy for setting the log wage based on the information it has about the worker, we use the Kalman filter described in this quantecon lecture *A First Look at the Kalman filter*.

In particular, we want to compute all of the objects in an "innovation representation".

2.4 An Innovations Representation

We have all the objects in hand required to form an innovations representation for the output process $\{y_t\}_{t=0}^T$ for a worker.

Let's code that up now.

$$\begin{aligned} \hat{x}_{t+1} &= A\hat{x}_t + K_t a_t \\ y_t &= G\hat{x}_t + a_t \end{aligned}$$

where K_t is the Kalman gain matrix at time t.

We accomplish this in the following code that uses the `Kalman` class.

```python
kalman = Kalman(ss, xhat_0, Σ_0)
Σ_t = np.zeros((*Σ_0.shape, T-1))
y_hat_t = np.zeros(T-1)
x_hat_t = np.zeros((2, T-1))

for t in range(1, T):
    kalman.update(y[t])
    x_hat, Σ = kalman.x_hat, kalman.Sigma
    Σ_t[:, :, t-1] = Σ
    x_hat_t[:, t-1] = x_hat.reshape(-1)
    y_hat_t[t-1] = worker.G @ x_hat

x_hat_t = np.concatenate((x[:, 1][:, np.newaxis],
                    x_hat_t), axis=1)
Σ_t = np.concatenate((worker.Σ_0[:, :, np.newaxis],
                    Σ_t), axis=2)
u_hat_t = x_hat_t[1, :]
```

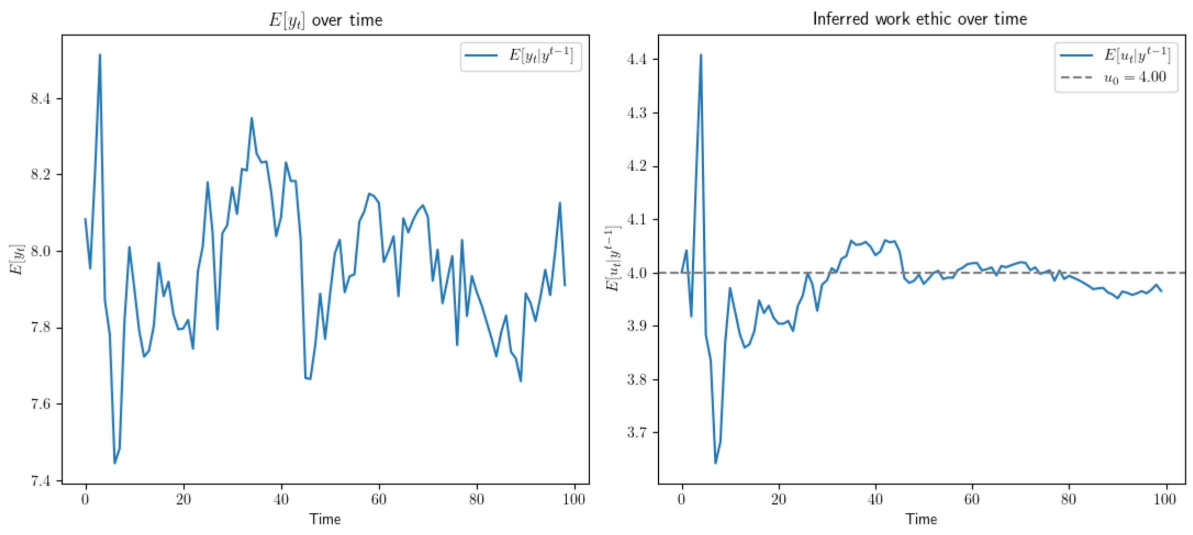
```
/tmp/ipykernel_6165/2927621375.py:11: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  y_hat_t[t-1] = worker.G @ x_hat
```

For a draw of h_0, u_0, we plot $Ey_t = G\hat{x}_t$ where $\hat{x}_t = E[x_t|y^{t-1}]$.

We also plot $E[u_0|y^{t-1}]$, which is the firm inference about a worker's hard-wired "work ethic" u_0, conditioned on information y^{t-1} that it has about him or her coming into period t.

We can watch as the firm's inference $E[u_0|y^{t-1}]$ of the worker's work ethic converges toward the hidden u_0, which is not directly observed by the firm.

```
fig, ax = plt.subplots(1, 2)

ax[0].plot(y_hat_t, label=r'$E[y_t| y^{t-1}]$')
ax[0].set_xlabel('Time')
ax[0].set_ylabel(r'$E[y_t]$')
ax[0].set_title(r'$E[y_t]$ over time')
ax[0].legend()

ax[1].plot(u_hat_t, label=r'$E[u_t|y^{t-1}]$')
ax[1].axhline(y=u_0, color='grey',
            linestyle='dashed', label=fr'$u_0={u_0:.2f}$')
ax[1].set_xlabel('Time')
ax[1].set_ylabel(r'$E[u_t|y^{t-1}]$')
ax[1].set_title('Inferred work ethic over time')
ax[1].legend()

fig.tight_layout()
plt.show()
```



2.5 Some Computational Experiments

Let's look at Σ_0 and Σ_T in order to see how much the firm learns about the hidden state during the horizon we have set.

```
print(Σ_t[:, :, 0])
```

```
[[4. 0.]
 [0. 4.]]
```

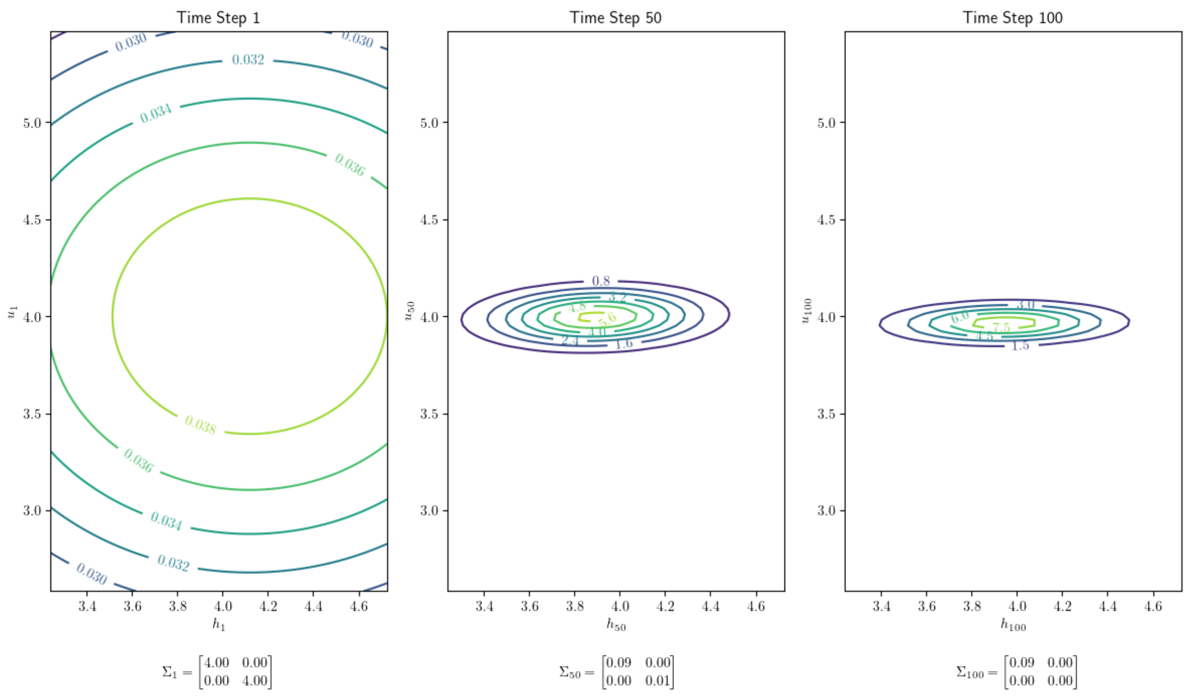
```
print(Σ_t[:, :, -1])
```

```
[[0.08805027 0.00100377]
 [0.00100377 0.00398351]]
```

Evidently, entries in the conditional covariance matrix become smaller over time.

It is enlightening to portray how conditional covariance matrices Σ_t evolve by plotting confidence ellipsoides around $E[x_t|y^{t-1}]$ at various t's.

```
# Create a grid of points for contour plotting
h_range = np.linspace(x_hat_t[0, :].min()-0.5*Σ_t[0, 0, 1],
                      x_hat_t[0, :].max()+0.5*Σ_t[0, 0, 1], 100)
u_range = np.linspace(x_hat_t[1, :].min()-0.5*Σ_t[1, 1, 1],
                      x_hat_t[1, :].max()+0.5*Σ_t[1, 1, 1], 100)
h, u = np.meshgrid(h_range, u_range)

# Create a figure with subplots for each time step
fig, axs = plt.subplots(1, 3, figsize=(12, 7))

# Iterate through each time step
for i, t in enumerate(np.linspace(0, T-1, 3, dtype=int)):
    # Create a multivariate normal distribution with x_hat and Σ at time step t
    mu = x_hat_t[:, t]
    cov = Σ_t[:, :, t]
    mvn = multivariate_normal(mean=mu, cov=cov)

    # Evaluate the multivariate normal PDF on the grid
    pdf_values = mvn.pdf(np.dstack((h, u)))

    # Create a contour plot for the PDF
    con = axs[i].contour(h, u, pdf_values, cmap='viridis')
    axs[i].clabel(con, inline=1, fontsize=10)
    axs[i].set_title(f'Time Step {t+1}')
    axs[i].set_xlabel(r'$h_{{{}}}$'.format(str(t+1)))
    axs[i].set_ylabel(r'$u_{{{}}}$'.format(str(t+1)))

    cov_latex = r'$\Sigma_{{{}}}= \begin{{bmatrix}} {:.2f} & {:.2f} \\ {:.2f} & {:.2f}
↪ \end{{bmatrix}}$'.format(
        t+1, cov[0, 0], cov[0, 1], cov[1, 0], cov[1, 1]
    )
    axs[i].text(0.33, -0.15, cov_latex, transform=axs[i].transAxes)


plt.tight_layout()
plt.show()
```



Note how the accumulation of evidence y^t affects the shape of the confidence ellipsoid as sample size t grows.

Now let's use our code to set the hidden state x_0 to a particular vector in order to watch how a firm learns starting from some x_0 we are interested in.

For example, let's say $h_0 = 0$ and $u_0 = 4$.

Here is one way to do this.

```python
# For example, we might want h_0 = 0 and u_0 = 4
mu_0 = np.array([0.0, 4.0])

# Create a LinearStateSpace object with Sigma_0 as a matrix of zeros
ss_example = LinearStateSpace(A, C, G, np.sqrt(R), mu_0=mu_0,
                              # This line forces exact h_0=0 and u_0=4
                              Sigma_0=np.zeros((2, 2))
                             )

T = 100
x, y = ss_example.simulate(T)
y = y.flatten()

# Now h_0=0 and u_0=4
h_0, u_0 = x[0, 0], x[1, 0]
print('h_0 =', h_0)
print('u_0 =', u_0)
```

```
h_0 = 0.0
u_0 = 4.0
```

Another way to accomplish the same goal is to use the following code.

```python
# If we want to set the initial
# h_0 = hhat_0 = 0 and u_0 = uhhat_0 = 4.0:
worker = create_worker(hhat_0=0.0, uhat_0=4.0)

ss_example = LinearStateSpace(A, C, G, np.sqrt(R),
                              # This line takes h_0=hhat_0 and u_0=uhhat_0
                              mu_0=worker.xhat_0,
                              # This line forces exact h_0=hhat_0 and u_0=uhhat_0
                              Sigma_0=np.zeros((2, 2))
                             )

T = 100
x, y = ss_example.simulate(T)
y = y.flatten()

# Now h_0 and u_0 will be exactly hhat_0
h_0, u_0 = x[0, 0], x[1, 0]
print('h_0 =', h_0)
print('u_0 =', u_0)
```

```
h_0 = 0.0
u_0 = 4.0
```

For this worker, let's generate a plot like the one above.

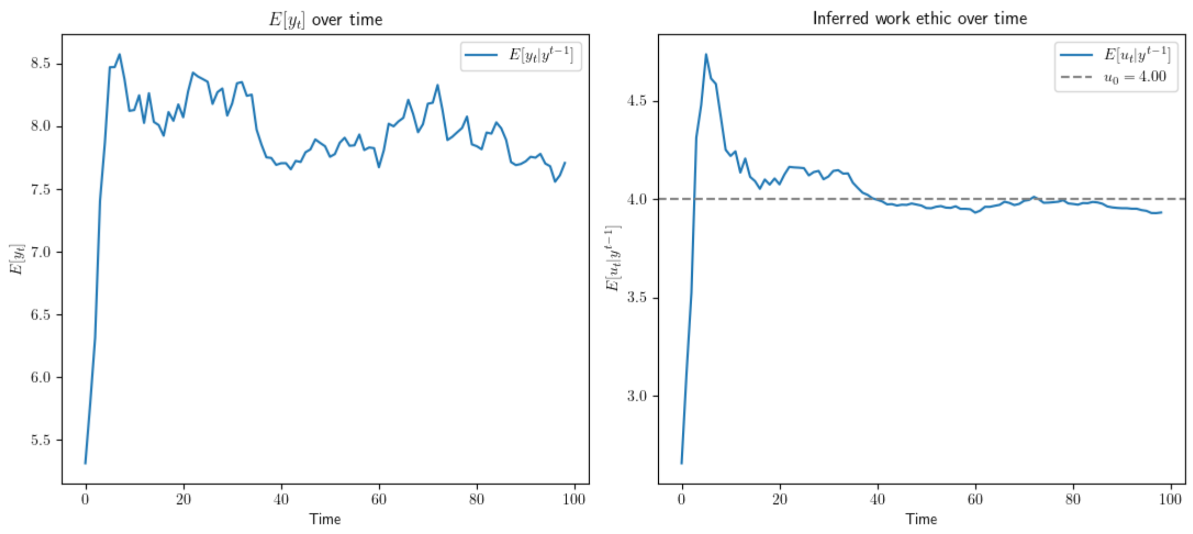
```python
# First we compute the Kalman filter with initial xhat_0 and Σ_0
kalman = Kalman(ss, xhat_0, Σ_0)
Σ_t = []
y_hat_t = np.zeros(T-1)
u_hat_t = np.zeros(T-1)

# Then we iteratively update the Kalman filter class using
# observation y based on the linear state model above:
for t in range(1, T):
    kalman.update(y[t])
    x_hat, Σ = kalman.x_hat, kalman.Sigma
    Σ_t.append(Σ)
    y_hat_t[t-1] = worker.G @ x_hat
    u_hat_t[t-1] = x_hat[1]


# Generate plots for y_hat_t and u_hat_t
fig, ax = plt.subplots(1, 2)

ax[0].plot(y_hat_t, label=r'$E[y_t| y^{t-1}]$')
ax[0].set_xlabel('Time')
ax[0].set_ylabel(r'$E[y_t]$')
ax[0].set_title(r'$E[y_t]$ over time')
ax[0].legend()

ax[1].plot(u_hat_t, label=r'$E[u_t|y^{t-1}]$')
ax[1].axhline(y=u_0, color='grey',
            linestyle='dashed', label=fr'$u_0={u_0:.2f}$')
ax[1].set_xlabel('Time')
ax[1].set_ylabel(r'$E[u_t|y^{t-1}]$')
ax[1].set_title('Inferred work ethic over time')
```

(continues on next page)

(continued from previous page)

```
ax[1].legend()

fig.tight_layout()
plt.show()
```

```
/tmp/ipykernel_6165/1462412779.py:13: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  y_hat_t[t-1] = worker.G @ x_hat
/tmp/ipykernel_6165/1462412779.py:14: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  u_hat_t[t-1] = x_hat[1]
```



More generally, we can change some or all of the parameters defining a worker in our `create_worker` namedtuple.

Here is an example.

```
# We can set these parameters when creating a worker -- just like classes!
hard_working_worker =  create_worker(α=.4, β=.8,
                        hhat_0=7.0, uhat_0=100, σ_h=2.5, σ_u=3.2)

print(hard_working_worker)
```

```
WorkerModel(A=array([[0.4, 0.8],
       [0. , 1. ]]), C=array([[0.2],
       [0. ]]), G=array([1., 1.]), R=0.5, xhat_0=array([[  7.],
       [100.]]), Σ_0=array([[2.5, 0. ],
       [0. , 3.2]]))
```

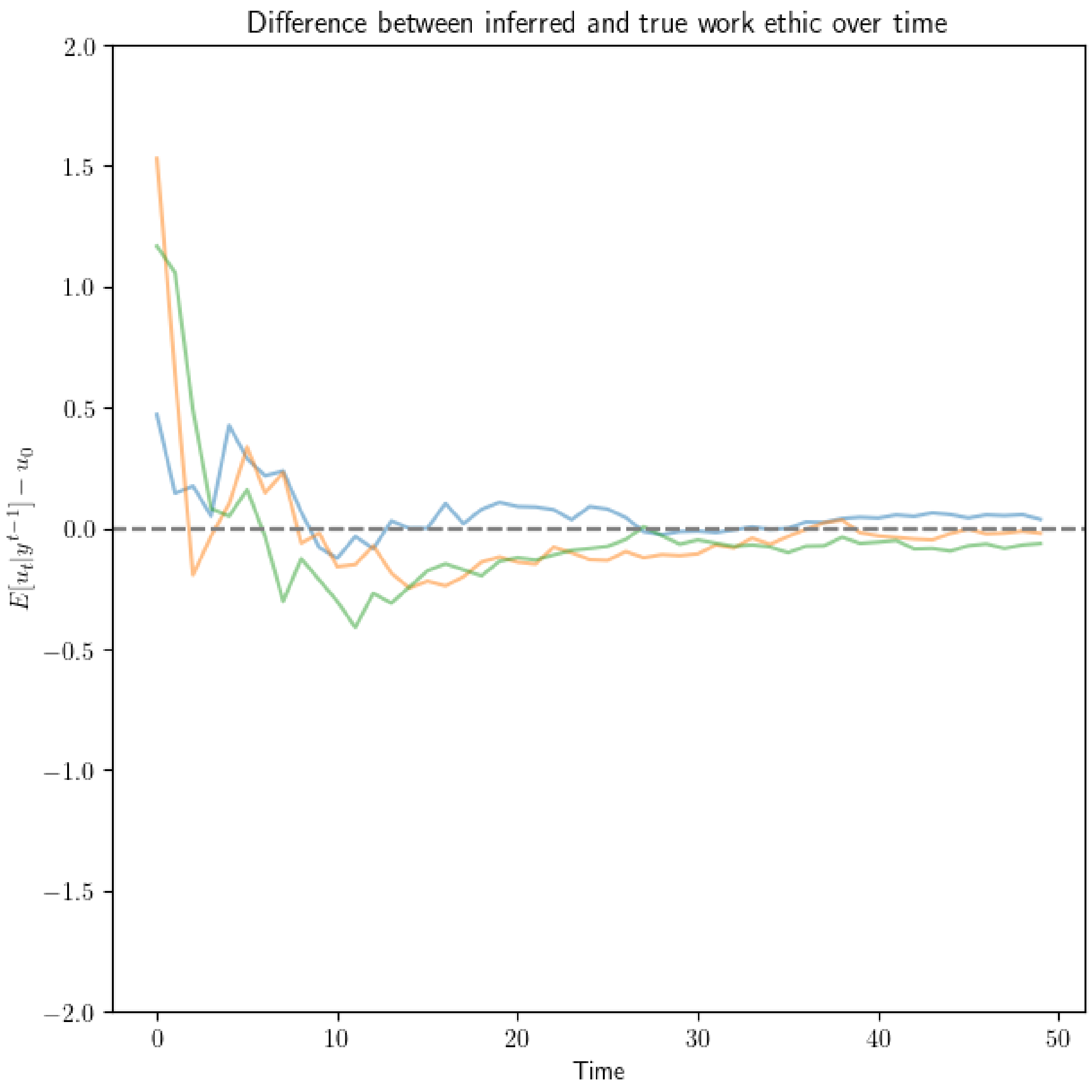
We can also simulate the system for $T = 50$ periods for different workers.

The difference between the inferred work ethics and true work ethics converges to 0 over time.

This shows that the filter is gradually teaching the worker and firm about the worker's effort.

```
num_workers = 3
T = 50
fig, ax = plt.subplots(figsize=(7, 7))

for i in range(num_workers):
    worker = create_worker(uhat_0=4+2*i)
    simulate_workers(worker, T, ax)
ax.set_ylim(ymin=-2, ymax=2)
plt.show()
```

```
/tmp/ipykernel_6165/2747793518.py:30: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  y_hat_t[i] = worker.G @ x_hat
/tmp/ipykernel_6165/2747793518.py:31: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  u_hat_t[i] = x_hat[1]
```



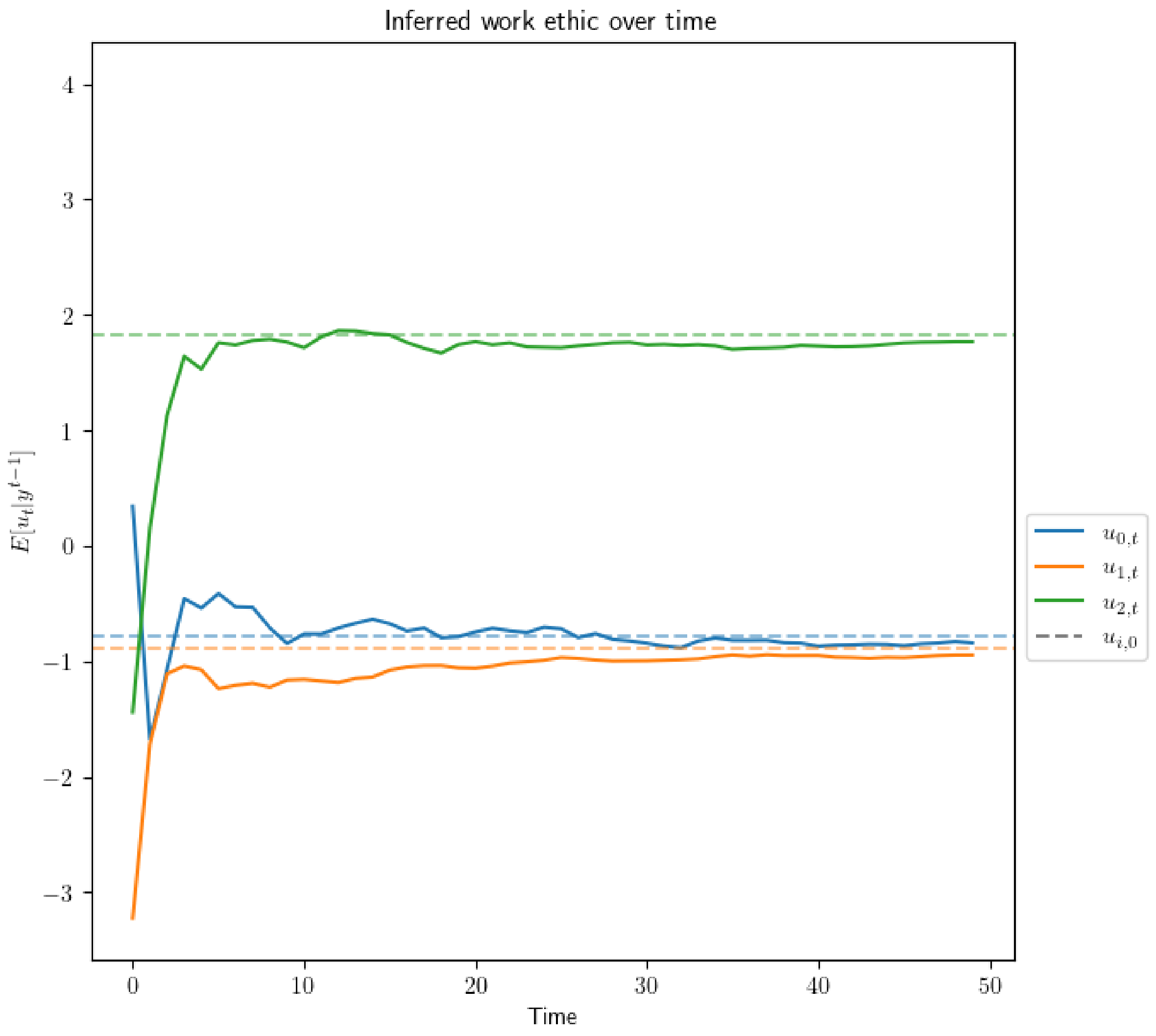
```
# We can also generate plots of u_t:

T = 50
fig, ax = plt.subplots(figsize=(7, 7))

uhat_0s = [2, -2, 1]
αs = [0.2, 0.3, 0.5]
βs = [0.1, 0.9, 0.3]

for i, (uhat_0, α, β) in enumerate(zip(uhat_0s, αs, βs)):
    worker = create_worker(uhat_0=uhat_0, α=α, β=β)
    simulate_workers(worker, T, ax,
                    # By setting diff=False, it will give u_t
                    diff=False, name=r'$u_{{{}, t}}$'.format(i))

ax.axhline(y=u_0, xmin=0, xmax=0, color='grey',
```

(continues on next page)

(continued from previous page)

```
            linestyle='dashed', label=r'$u_{i, 0}$')
ax.legend(bbox_to_anchor=(1, 0.5))
plt.show()
```

```
/tmp/ipykernel_6165/2747793518.py:30: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  y_hat_t[i] = worker.G @ x_hat
/tmp/ipykernel_6165/2747793518.py:31: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  u_hat_t[i] = x_hat[1]
```



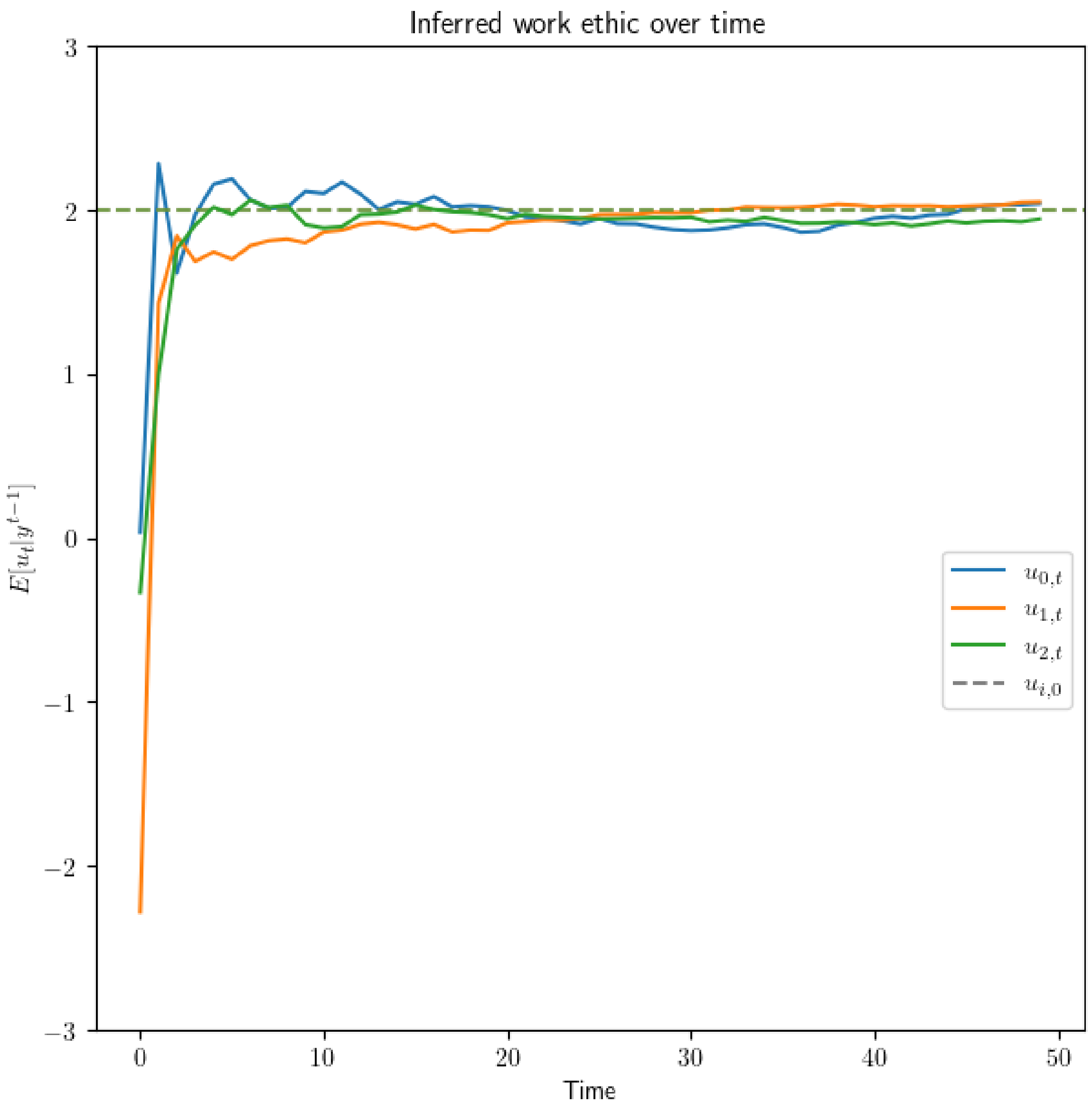
```
# We can also use exact u_0=1 and h_0=2 for all workers
```

(continues on next page)

(continued from previous page)

```
T = 50
fig, ax = plt.subplots(figsize=(7, 7))

# These two lines set u_0=1 and h_0=2 for all workers
mu_0 = np.array([[1],
                 [2]])
Sigma_0 = np.zeros((2,2))

uhat_0s = [2, -2, 1]
αs = [0.2, 0.3, 0.5]
βs = [0.1, 0.9, 0.3]

for i, (uhat_0, α, β) in enumerate(zip(uhat_0s, αs, βs)):
    worker = create_worker(uhat_0=uhat_0, α=α, β=β)
    simulate_workers(worker, T, ax, mu_0=mu_0, Sigma_0=Sigma_0,
                     diff=False, name=r'$u_{{{}, t}}$'.format(i))

# This controls the boundary of plots
ax.set_ylim(ymin=-3, ymax=3)
ax.axhline(y=u_0, xmin=0, xmax=0, color='grey',
           linestyle='dashed', label=r'$u_{i, 0}$')
ax.legend(bbox_to_anchor=(1, 0.5))
plt.show()
```

```
/tmp/ipykernel_6165/2747793518.py:30: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  y_hat_t[i] = worker.G @ x_hat
/tmp/ipykernel_6165/2747793518.py:31: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  u_hat_t[i] = x_hat[1]
```



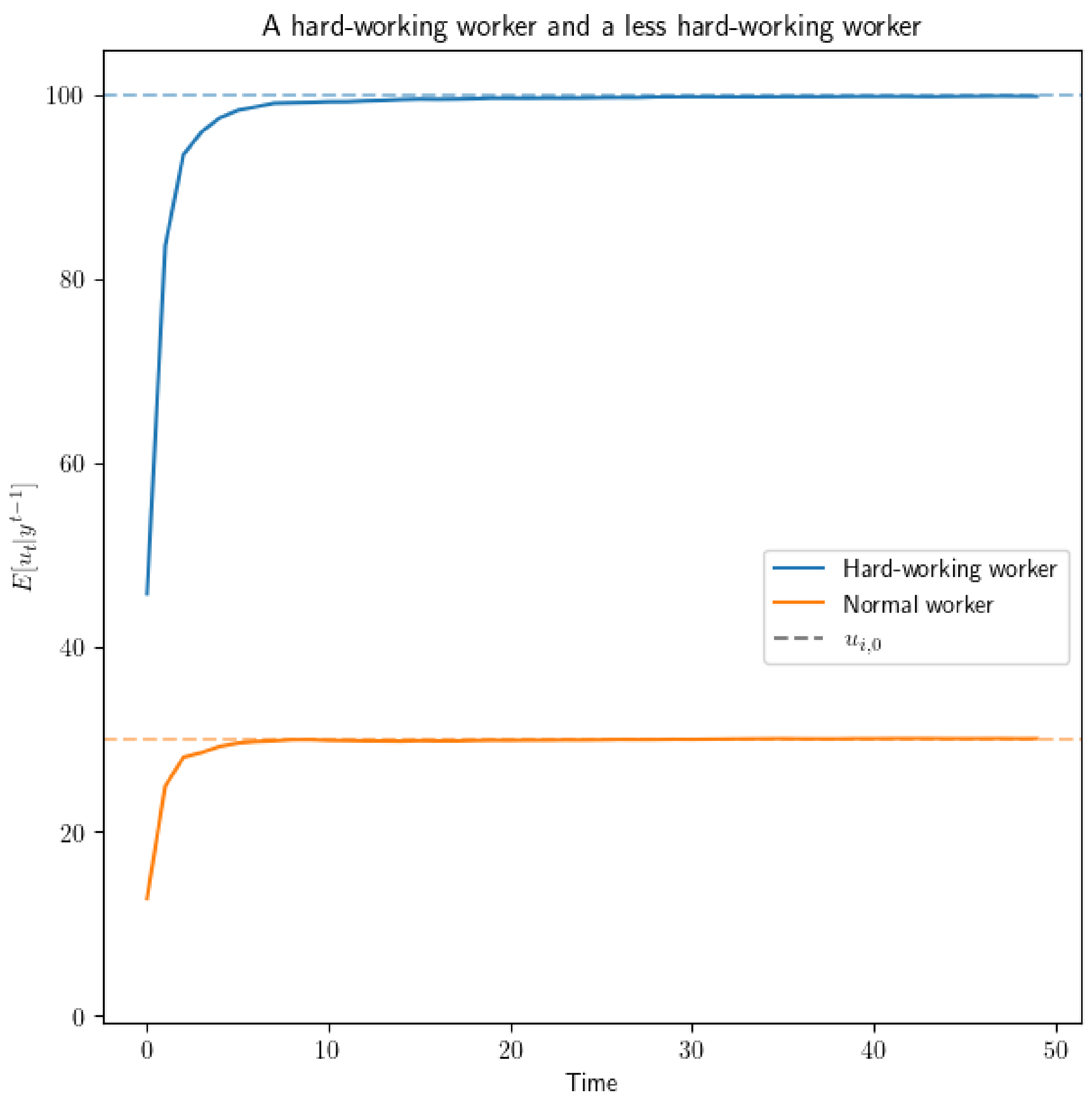
```
# We can generate a plot for only one of the workers:

T = 50
fig, ax = plt.subplots(figsize=(7, 7))

mu_0_1 = np.array([[1],
                   [100]])
mu_0_2 = np.array([[1],
                   [30]])
Sigma_0 = np.zeros((2,2))

uhat_0s = 100
αs = 0.5
βs = 0.3

worker = create_worker(uhat_0=uhat_0, α=α, β=β)
```

(continues on next page)

(continued from previous page)

```
simulate_workers(worker, T, ax, mu_0=mu_0_1, Sigma_0=Sigma_0,
                 diff=False, name=r'Hard-working worker')
simulate_workers(worker, T, ax, mu_0=mu_0_2, Sigma_0=Sigma_0,
                 diff=False,
                 title='A hard-working worker and a less hard-working worker',
                 name=r'Normal worker')
ax.axhline(y=u_0, xmin=0, xmax=0, color='grey',
           linestyle='dashed', label=r'$u_{i, 0}$')
ax.legend(bbox_to_anchor=(1, 0.5))
plt.show()
```

```
/tmp/ipykernel_6165/2747793518.py:30: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  y_hat_t[i] = worker.G @ x_hat
/tmp/ipykernel_6165/2747793518.py:31: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  u_hat_t[i] = x_hat[1]
```



2.6 Future Extensions

We can do lots of enlightening experiments by creating new types of workers and letting the firm learn about their hidden (to the firm) states by observing just their output histories.

CHAPTER

THREE

REVERSE ENGINEERING A LA MUTH

Contents

In addition to what's in Anaconda, this lecture uses the quantecon library.

```
!pip install --upgrade quantecon
```

We'll also need the following imports:

```
import matplotlib.pyplot as plt
import numpy as np

from quantecon import Kalman
from quantecon import LinearStateSpace
np.set_printoptions(linewidth=120, precision=4, suppress=True)
```

This lecture uses the Kalman filter to reformulate John F. Muth's first paper [Muth, 1960] about rational expectations.

Muth used *classical* prediction methods to reverse engineer a stochastic process that renders optimal Milton Friedman's [Friedman, 1956] "adaptive expectations" scheme.

3.1 Friedman (1956) and Muth (1960)

Milton Friedman [Friedman, 1956] (1956) posited that consumer's forecast their future disposable income with the adaptive expectations scheme

$$y^*_{t+i,t} = K \sum_{j=0}^{\infty} (1-K)^j y_{t-j} \tag{3.1}$$

where $K \in (0,1)$ and $y^*_{t+i,t}$ is a forecast of future y over horizon i.

Milton Friedman justified the **exponential smoothing** forecasting scheme (3.1) informally, noting that it seemed a plausible way to use past income to forecast future income.

In his first paper about rational expectations, John F. Muth [Muth, 1960] reverse-engineered a univariate stochastic process $\{y_t\}_{t=-\infty}^{\infty}$ for which Milton Friedman's adaptive expectations scheme gives linear least forecasts of y_{t+j} for any horizon i.

Muth sought a setting and a sense in which Friedman's forecasting scheme is optimal.

That is, Muth asked for what optimal forecasting **question** is Milton Friedman's adaptive expectation scheme the **answer**.

Muth (1960) used classical prediction methods based on lag-operators and z-transforms to find the answer to his question.

Please see lectures Classical Control with Linear Algebra and Classical Filtering and Prediction with Linear Algebra for an introduction to the classical tools that Muth used.

Rather than using those classical tools, in this lecture we apply the Kalman filter to express the heart of Muth's analysis concisely.

The lecture First Look at Kalman Filter describes the Kalman filter.

We'll use limiting versions of the Kalman filter corresponding to what are called **stationary values** in that lecture.

3.2 A Process for Which Adaptive Expectations are Optimal

Suppose that an observable y_t is the sum of an unobserved random walk x_t and an IID shock $\epsilon_{2,t}$:

$$\begin{aligned} x_{t+1} &= x_t + \sigma_x \epsilon_{1,t+1} \\ y_t &= x_t + \sigma_y \epsilon_{2,t} \end{aligned} \tag{3.2}$$

where

$$\begin{bmatrix} \epsilon_{1,t+1} \\ \epsilon_{2,t} \end{bmatrix} \sim \mathcal{N}(0, I)$$

is an IID process.

Note: A property of the state-space representation (3.2) is that in general neither $\epsilon_{1,t}$ nor $\epsilon_{2,t}$ is in the space spanned by square-summable linear combinations of $y_t, y_{t-1}, \ldots$.

In general $\begin{bmatrix} \epsilon_{1,t} \\ \epsilon_{2t} \end{bmatrix}$ has more information about future y_{t+j}'s than is contained in $y_t, y_{t-1}, \ldots$.

We can use the asymptotic or stationary values of the Kalman gain and the one-step-ahead conditional state covariance matrix to compute a time-invariant *innovations representation*

$$\begin{aligned} \hat{x}_{t+1} &= \hat{x}_t + K a_t \\ y_t &= \hat{x}_t + a_t \end{aligned} \tag{3.3}$$

where $\hat{x}_t = E[x_t|y_{t-1}, y_{t-2}, ...]$ and $a_t = y_t - E[y_t|y_{t-1}, y_{t-2}, ...]$.

Note: A key property about an *innovations representation* is that a_t is in the space spanned by square summable linear combinations of $y_t, y_{t-1}, ...$.

For more ramifications of this property, see the lectures *Shock Non-Invertibility* and *Recursive Models of Dynamic Linear Economies*.

Later we'll stack these state-space systems (3.2) and (3.3) to display some classic findings of Muth.

But first, let's create an instance of the state-space system (3.2) then apply the quantecon `Kalman` class, then uses it to construct the associated "innovations representation"

```
# Make some parameter choices
# sigx/sigy are state noise std err and measurement noise std err
μ_0, σ_x, σ_y = 10, 1, 5

# Create a LinearStateSpace object
A, C, G, H = 1, σ_x, 1, σ_y
ss = LinearStateSpace(A, C, G, H, mu_0=μ_0)

# Set prior and initialize the Kalman type
x_hat_0, Σ_0 = 10, 1
kmuth = Kalman(ss, x_hat_0, Σ_0)

# Computes stationary values which we need for the innovation
# representation
S1, K1 = kmuth.stationary_values()

# Extract scalars from nested arrays
S1, K1 = S1.item(), K1.item()

# Form innovation representation state-space
Ak, Ck, Gk, Hk = A, K1, G, 1

ssk = LinearStateSpace(Ak, Ck, Gk, Hk, mu_0=x_hat_0)
```

3.3 Some Useful State-Space Math

Now we want to map the time-invariant innovations representation (3.3) and the original state-space system (3.2) into a convenient form for deducing the impulse responses from the original shocks to the x_t and $\hat{x}_t$.

Putting both of these representations into a single state-space system is yet another application of the insight that "finding the state is an art".

We'll define a state vector and appropriate state-space matrices that allow us to represent both systems in one fell swoop.

Note that

$$a_t = x_t + \sigma_y \epsilon_{2,t} - \hat{x}_t$$

so that

$$\begin{aligned}\hat{x}_{t+1} &= \hat{x}_t + K(x_t + \sigma_y \epsilon_{2,t} - \hat{x}_t)\\ &= (1-K)\hat{x}_t + Kx_t + K\sigma_y \epsilon_{2,t}\end{aligned}$$

The stacked system

$$\begin{bmatrix} x_{t+1} \\ \hat{x}_{t+1} \\ \epsilon_{2,t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ K & (1-K) & K\sigma_y \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ \hat{x}_t \\ \epsilon_{2,t} \end{bmatrix} + \begin{bmatrix} \sigma_x & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \epsilon_{1,t+1} \\ \epsilon_{2,t+1} \end{bmatrix}$$

$$\begin{bmatrix} y_t \\ a_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & \sigma_y \\ 1 & -1 & \sigma_y \end{bmatrix} \begin{bmatrix} x_t \\ \hat{x}_t \\ \epsilon_{2,t} \end{bmatrix}$$

is a state-space system that tells us how the shocks $\begin{bmatrix} \epsilon_{1,t+1} \\ \epsilon_{2,t+1} \end{bmatrix}$ affect states $\hat{x}_{t+1}, x_t$, the observable y_t, and the innovation a_t.

With this tool at our disposal, let's form the composite system and simulate it

```
# Create grand state-space for y_t, a_t as observed vars -- Use
# stacking trick above
Af = np.array([[ 1,      0,        0],
               [K1, 1 - K1, K1 * σ_y],
               [ 0,      0,        0]])
Cf = np.array([[σ_x,        0],
               [  0, K1 * σ_y],
               [  0,        1]])
Gf = np.array([[1,  0, σ_y],
               [1, -1, σ_y]])

μ_true, μ_prior = 10, 10
μ_f = np.array([μ_true, μ_prior, 0]).reshape(3, 1)

# Create the state-space
ssf = LinearStateSpace(Af, Cf, Gf, mu_0=μ_f)

# Draw observations of y from the state-space model
N = 50
xf, yf = ssf.simulate(N)

print(f"Kalman gain = {K1}")
print(f"Conditional variance = {S1}")
```

```
Kalman gain = 0.1809975124224177
Conditional variance = 5.524937810560442
```

Now that we have simulated our joint system, we have $x_t$, $\hat{x}_t$, and y_t.

We can now investigate how these variables are related by plotting some key objects.

3.4 Estimates of Unobservables

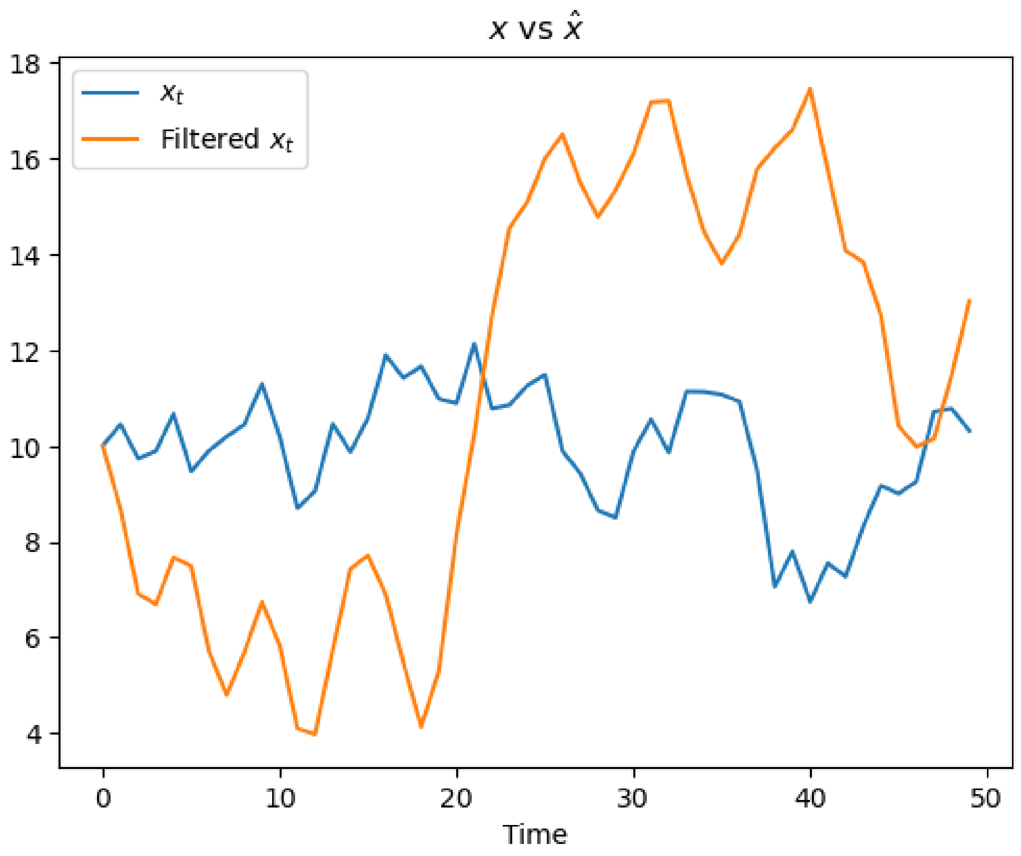
First, let's plot the hidden state $x_t$ and the filtered version $\hat{x}_t$ that is linear-least squares projection of $x_t$ on the history $y_{t-1}, y_{t-2}, \cdots$

```
fig, ax = plt.subplots()
ax.plot(xf[0, :], label="$x_t$")
ax.plot(xf[1, :], label="Filtered $x_t$")
```

(continues on next page)

(continued from previous page)

```
ax.legend()
ax.set_xlabel("Time")
ax.set_title(r"$x$ vs $\hat{x}$")
plt.show()
```



Note how x_t and $\hat{x}_t$ differ.

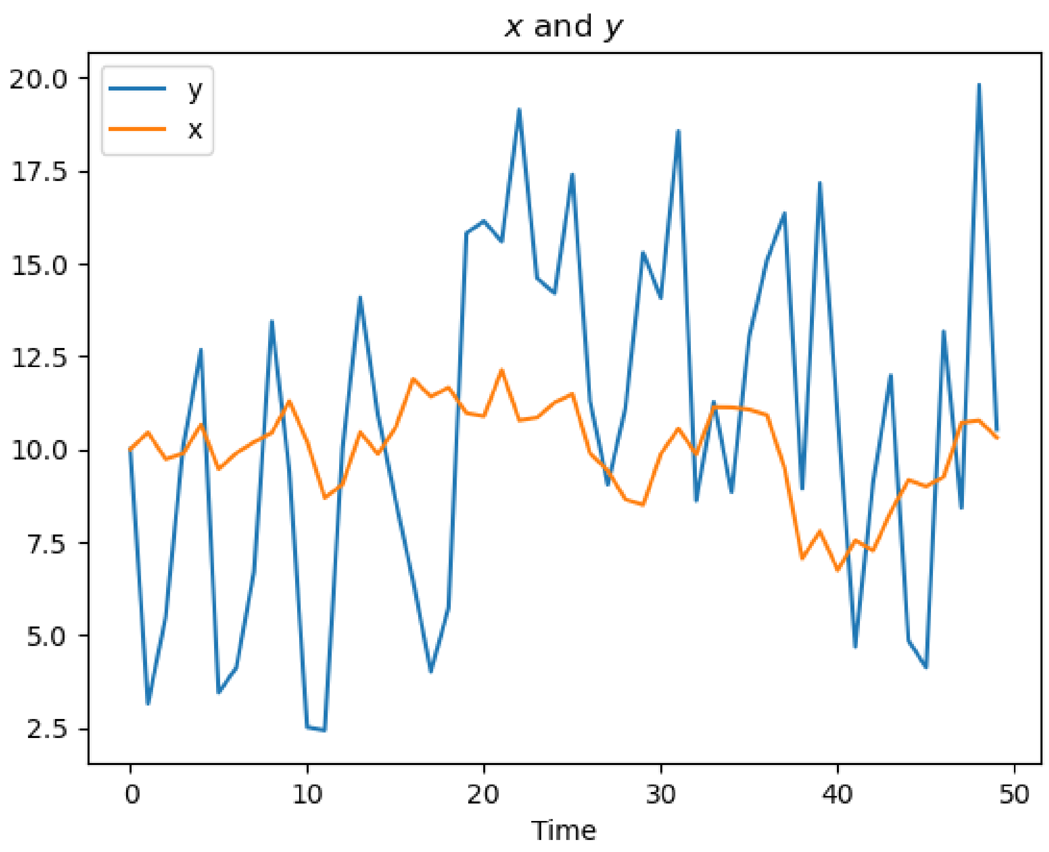
For Friedman, $\hat{x}_t$ and not x_t is the consumer's idea about her/his *permanent income*.

3.5 Relationship of Unobservables to Observables

Now let's plot x_t and y_t.

Recall that y_t is just x_t plus white noise

```
fig, ax = plt.subplots()
ax.plot(yf[0, :], label="y")
ax.plot(xf[0, :], label="x")
ax.legend()
ax.set_title(r"$x$ and $y$")
ax.set_xlabel("Time")
plt.show()
```

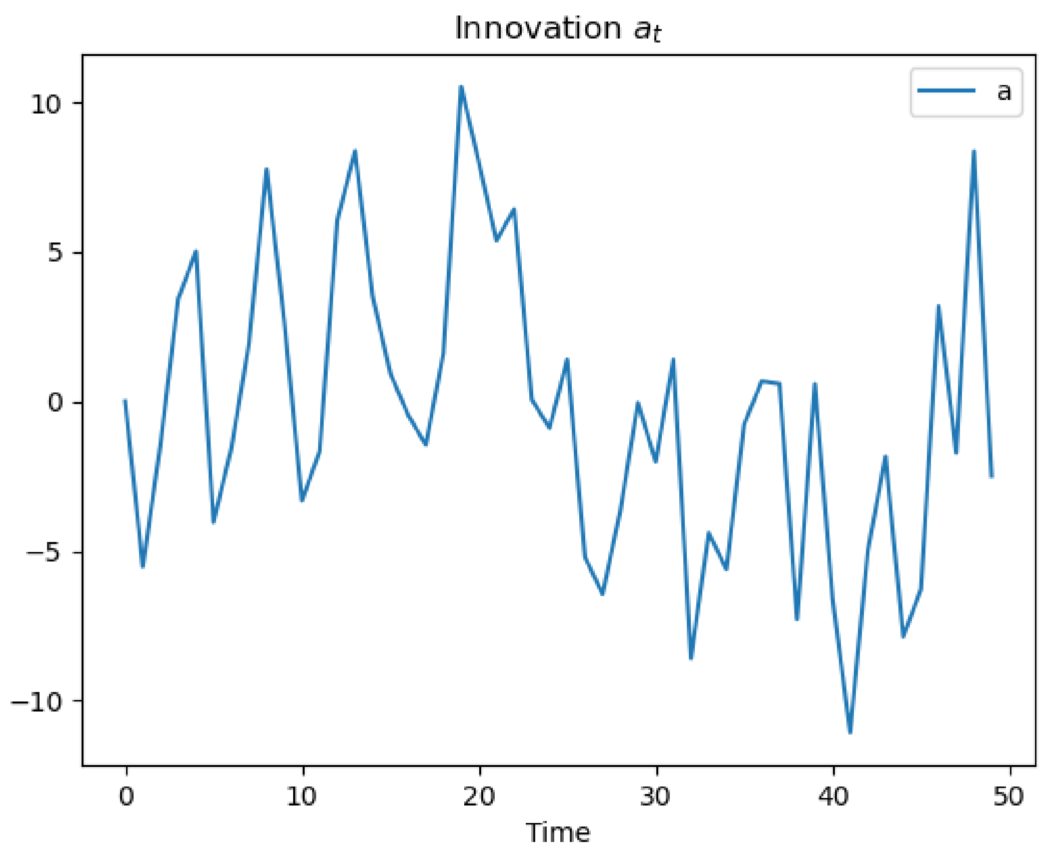


We see above that y seems to look like white noise around the values of x.

3.5.1 Innovations

Recall that we wrote down the innovation representation that depended on a_t. We now plot the innovations $\{a_t\}$:

```
fig, ax = plt.subplots()
ax.plot(yf[1, :], label="a")
ax.legend()
ax.set_title(r"Innovation $a_t$")
ax.set_xlabel("Time")
plt.show()
```



3.6 MA and AR Representations

Now we shall extract from the `Kalman` instance `kmuth` coefficients of

- a fundamental moving average representation that represents y_t as a one-sided moving sum of current and past a_ts that are square summable linear combinations of $y_t, y_{t-1}, \ldots$.
- a univariate autoregression representation that depicts the coefficients in a linear least square projection of y_t on the semi-infinite history $y_{t-1}, y_{t-2}, \ldots$.

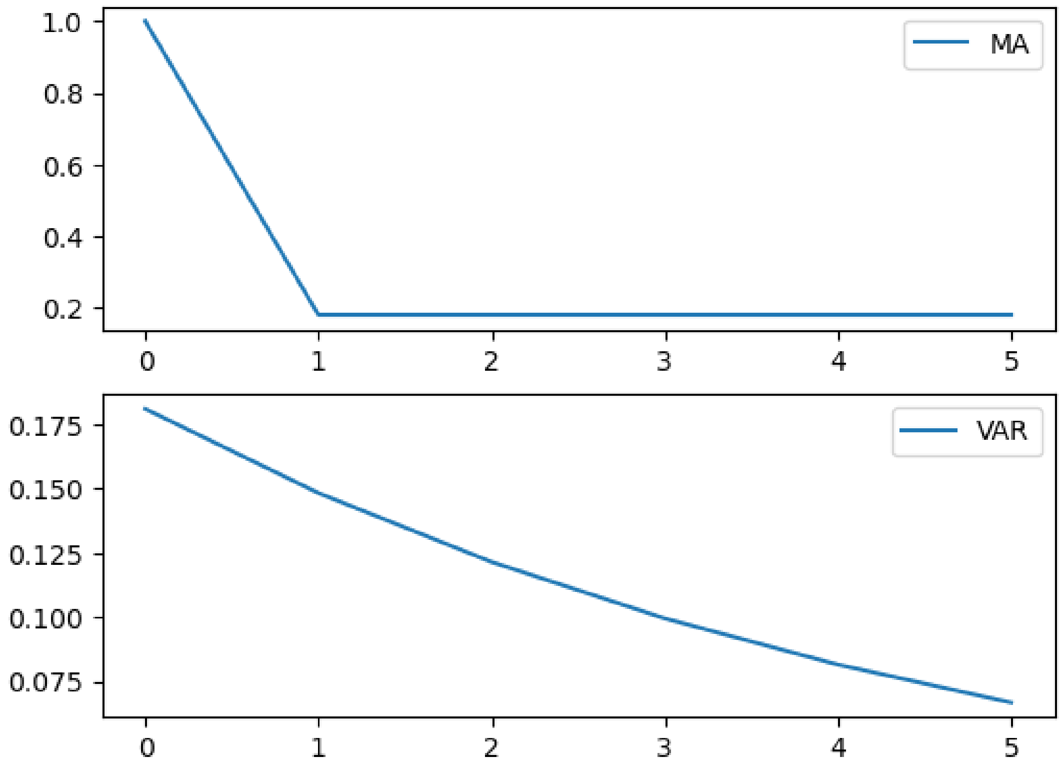
Then we'll plot each of them

```
# Kalman Methods for MA and VAR
coefs_ma = kmuth.stationary_coefficients(5, "ma")
coefs_var = kmuth.stationary_coefficients(5, "var")

# Coefficients come in a list of arrays, but we
# want to plot them and so need to stack into an array
coefs_ma_array = np.vstack(coefs_ma)
coefs_var_array = np.vstack(coefs_var)

fig, ax = plt.subplots(2)
ax[0].plot(coefs_ma_array, label="MA")
ax[0].legend()
ax[1].plot(coefs_var_array, label="VAR")
```

(continues on next page)

(continued from previous page)

```
ax[1].legend()

plt.show()
```



The **moving average** coefficients in the top panel show tell-tale signs of y_t being a process whose first difference is a first-order autoregression.

The **autoregressive coefficients** decline geometrically with decay rate $(1 - K)$.

These are exactly the target outcomes that Muth (1960) aimed to reverse engineer

```
print(f'decay parameter 1 - K1 = {1 - K1}')
```

```
decay parameter 1 - K1 = 0.8190024875775823
```

Part II

LQ Control

CHAPTER

FOUR

LQ CONTROL: FOUNDATIONS

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install quantecon
```

4.1 Overview

Linear quadratic (LQ) control refers to a class of dynamic optimization problems that have found applications in almost every scientific field.

This lecture provides an introduction to LQ control and its economic applications.

As we will see, LQ systems have a simple structure that makes them an excellent workhorse for a wide variety of economic problems.

Moreover, while the linear-quadratic structure is restrictive, it is in fact far more flexible than it may appear initially.

These themes appear repeatedly below.

Mathematically, LQ control problems are closely related to *the Kalman filter*

- Recursive formulations of linear-quadratic control problems and Kalman filtering problems both involve matrix **Riccati equations**.
- Classical formulations of linear control and linear filtering problems make use of similar matrix decompositions (see for example this lecture and this lecture).

In reading what follows, it will be useful to have some familiarity with

- matrix manipulations
- vectors of random variables
- dynamic programming and the Bellman equation (see for example this lecture and this lecture)

For additional reading on LQ control, see, for example,

- [Ljungqvist and Sargent, 2018], chapter 5
- [Hansen and Sargent, 2008], chapter 4
- [Hernandez-Lerma and Lasserre, 1996], section 3.5

In order to focus on computation, we leave longer proofs to these sources (while trying to provide as much intuition as possible).

Let's start with some imports:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
from quantecon import LQ
```

4.2 Introduction

The "linear" part of LQ is a linear law of motion for the state, while the "quadratic" part refers to preferences.

Let's begin with the former, move on to the latter, and then put them together into an optimization problem.

4.2.1 The Law of Motion

Let x_t be a vector describing the state of some economic system.

Suppose that x_t follows a linear law of motion given by

$$x_{t+1} = Ax_t + Bu_t + Cw_{t+1}, \qquad t = 0, 1, 2, \ldots \tag{4.1}$$

Here

- u_t is a "control" vector, incorporating choices available to a decision-maker confronting the current state x_t
- $\{w_t\}$ is an uncorrelated zero mean shock process satisfying $\mathbb{E}w_t w_t' = I$, where the right-hand side is the identity matrix

Regarding the dimensions

- x_t is $n \times 1$, A is $n \times n$
- u_t is $k \times 1$, B is $n \times k$
- w_t is $j \times 1$, C is $n \times j$

Example 1

Consider a household budget constraint given by

$$a_{t+1} + c_t = (1+r)a_t + y_t$$

Here a_t is assets, r is a fixed interest rate, c_t is current consumption, and y_t is current non-financial income.

If we suppose that $\{y_t\}$ is serially uncorrelated and $N(0, \sigma^2)$, then, taking $\{w_t\}$ to be standard normal, we can write the system as

$$a_{t+1} = (1+r)a_t - c_t + \sigma w_{t+1}$$

This is clearly a special case of (4.1), with assets being the state and consumption being the control.

Example 2

One unrealistic feature of the previous model is that non-financial income has a zero mean and is often negative.

This can easily be overcome by adding a sufficiently large mean.

Hence in this example, we take $y_t = \sigma w_{t+1} + \mu$ for some positive real number μ.

Another alteration that's useful to introduce (we'll see why soon) is to change the control variable from consumption to the deviation of consumption from some "ideal" quantity $\bar{c}$.

(Most parameterizations will be such that $\bar{c}$ is large relative to the amount of consumption that is attainable in each period, and hence the household wants to increase consumption.)

For this reason, we now take our control to be $u_t := c_t - \bar{c}$.

In terms of these variables, the budget constraint $a_{t+1} = (1+r)a_t - c_t + y_t$ becomes

$$a_{t+1} = (1+r)a_t - u_t - \bar{c} + \sigma w_{t+1} + \mu \tag{4.2}$$

How can we write this new system in the form of equation (4.1)?

If, as in the previous example, we take a_t as the state, then we run into a problem: the law of motion contains some constant terms on the right-hand side.

This means that we are dealing with an *affine* function, not a linear one (recall this discussion).

Fortunately, we can easily circumvent this problem by adding an extra state variable.

In particular, if we write

$$\begin{pmatrix} a_{t+1} \\ 1 \end{pmatrix} = \begin{pmatrix} 1+r & -\bar{c}+\mu \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a_t \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \end{pmatrix} u_t + \begin{pmatrix} \sigma \\ 0 \end{pmatrix} w_{t+1} \tag{4.3}$$

then the first row is equivalent to (4.2).

Moreover, the model is now linear and can be written in the form of (4.1) by setting

$$x_t := \begin{pmatrix} a_t \\ 1 \end{pmatrix}, \quad A := \begin{pmatrix} 1+r & -\bar{c}+\mu \\ 0 & 1 \end{pmatrix}, \quad B := \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad C := \begin{pmatrix} \sigma \\ 0 \end{pmatrix} \tag{4.4}$$

In effect, we've bought ourselves linearity by adding another state.

4.2.2 Preferences

In the LQ model, the aim is to minimize flow of losses, where time-t loss is given by the quadratic expression

$$x_t' R x_t + u_t' Q u_t \tag{4.5}$$

Here

- R is assumed to be $n \times n$, symmetric and nonnegative definite.
- Q is assumed to be $k \times k$, symmetric and positive definite.

Note: In fact, for many economic problems, the definiteness conditions on R and Q can be relaxed. It is sufficient that certain submatrices of R and Q be nonnegative definite. See [Hansen and Sargent, 2008] for details.

Example 1

A very simple example that satisfies these assumptions is to take R and Q to be identity matrices so that current loss is

$$x_t' I x_t + u_t' I u_t = \|x_t\|^2 + \|u_t\|^2$$

Thus, for both the state and the control, loss is measured as squared distance from the origin.

(In fact, the general case (4.5) can also be understood in this way, but with R and Q identifying other – non-Euclidean – notions of "distance" from the zero vector.)

Intuitively, we can often think of the state x_t as representing deviation from a target, such as

- deviation of inflation from some target level
- deviation of a firm's capital stock from some desired quantity

The aim is to put the state close to the target, while using controls parsimoniously.

Example 2

In the household problem *studied above*, setting $R = 0$ and $Q = 1$ yields preferences

$$x_t' R x_t + u_t' Q u_t = u_t^2 = (c_t - \bar{c})^2$$

Under this specification, the household's current loss is the squared deviation of consumption from the ideal level $\bar{c}$.

4.3 Optimality – Finite Horizon

Let's now be precise about the optimization problem we wish to consider, and look at how to solve it.

4.3.1 The Objective

We will begin with the finite horizon case, with terminal time $T \in \mathbb{N}$.

In this case, the aim is to choose a sequence of controls $\{u_0, \dots, u_{T-1}\}$ to minimize the objective

$$\mathbb{E}\left\{\sum_{t=0}^{T-1} \beta^t (x_t' R x_t + u_t' Q u_t) + \beta^T x_T' R_f x_T\right\} \tag{4.6}$$

subject to the law of motion (4.1) and initial state x_0.

The new objects introduced here are β and the matrix R_f.

The scalar β is the discount factor, while $x' R_f x$ gives terminal loss associated with state x.

Comments:

- We assume R_f to be $n \times n$, symmetric and nonnegative definite.
- We allow $\beta = 1$, and hence include the undiscounted case.
- x_0 may itself be random, in which case we require it to be independent of the shock sequence $w_1, \dots, w_T$.

4.3.2 Information

There's one constraint we've neglected to mention so far, which is that the decision-maker who solves this LQ problem knows only the present and the past, not the future.

To clarify this point, consider the sequence of controls $\{u_0, \dots, u_{T-1}\}$.

When choosing these controls, the decision-maker is permitted to take into account the effects of the shocks $\{w_1, \dots, w_T\}$ on the system.

However, it is typically assumed — and will be assumed here — that the time-t control u_t can be made with knowledge of past and present shocks only.

The fancy measure-theoretic way of saying this is that u_t must be measurable with respect to the σ-algebra generated by $x_0, w_1, w_2, \dots, w_t$.

This is in fact equivalent to stating that u_t can be written in the form $u_t = g_t(x_0, w_1, w_2, \dots, w_t)$ for some Borel measurable function g_t.

(Just about every function that's useful for applications is Borel measurable, so, for the purposes of intuition, you can read that last phrase as "for some function g_t")

Now note that x_t will ultimately depend on the realizations of $x_0, w_1, w_2, \dots, w_t$.

In fact, it turns out that x_t summarizes all the information about these historical shocks that the decision-maker needs to set controls optimally.

More precisely, it can be shown that any optimal control u_t can always be written as a function of the current state alone.

Hence in what follows we restrict attention to control policies (i.e., functions) of the form $u_t = g_t(x_t)$.

Actually, the preceding discussion applies to all standard dynamic programming problems.

What's special about the LQ case is that – as we shall soon see — the optimal u_t turns out to be a linear function of x_t.

4.3.3 Solution

To solve the finite horizon LQ problem we can use a dynamic programming strategy based on backward induction that is conceptually similar to the approach adopted in this lecture.

For reasons that will soon become clear, we first introduce the notation $J_T(x) = x' R_f x$.

Now consider the problem of the decision-maker in the second to last period.

In particular, let the time be $T - 1$, and suppose that the state is x_{T-1}.

The decision-maker must trade-off current and (discounted) final losses, and hence solves

$$\min_u \{ x_{T-1}' R x_{T-1} + u' Q u + \beta \, \mathbb{E} J_T(A x_{T-1} + B u + C w_T) \}$$

At this stage, it is convenient to define the function

$$J_{T-1}(x) = \min_u \{ x' R x + u' Q u + \beta \, \mathbb{E} J_T(A x + B u + C w_T) \} \tag{4.7}$$

The function J_{T-1} will be called the $T-1$ value function, and $J_{T-1}(x)$ can be thought of as representing total "loss-to-go" from state x at time $T - 1$ when the decision-maker behaves optimally.

Now let's step back to $T - 2$.

For a decision-maker at $T-2$, the value $J_{T-1}(x)$ plays a role analogous to that played by the terminal loss $J_T(x) = x' R_f x$ for the decision-maker at $T - 1$.

That is, $J_{T-1}(x)$ summarizes the future loss associated with moving to state x.

The decision-maker chooses her control u to trade off current loss against future loss, where

- the next period state is $x_{T-1} = A x_{T-2} + B u + C w_{T-1}$, and hence depends on the choice of current control.
- the "cost" of landing in state x_{T-1} is $J_{T-1}(x_{T-1})$.

Her problem is therefore

$$\min_u \{ x_{T-2}' R x_{T-2} + u' Q u + \beta \, \mathbb{E} J_{T-1}(A x_{T-2} + B u + C w_{T-1}) \}$$

Letting

$$J_{T-2}(x) = \min_u \{ x' R x + u' Q u + \beta \, \mathbb{E} J_{T-1}(A x + B u + C w_{T-1}) \}$$

the pattern for backward induction is now clear.

In particular, we define a sequence of value functions $\{J_0, \dots, J_T\}$ via

$$J_{t-1}(x) = \min_u \{ x' R x + u' Q u + \beta \, \mathbb{E} J_t(A x + B u + C w_t) \} \quad \text{and} \quad J_T(x) = x' R_f x$$

The first equality is the Bellman equation from dynamic programming theory specialized to the finite horizon LQ problem.

Now that we have $\{J_0, \dots, J_T\}$, we can obtain the optimal controls.

As a first step, let's find out what the value functions look like.

It turns out that every J_t has the form $J_t(x) = x' P_t x + d_t$ where P_t is a $n \times n$ matrix and d_t is a constant.

We can show this by induction, starting from $P_T := R_f$ and $d_T = 0$.

Using this notation, (4.7) becomes

$$J_{T-1}(x) = \min_u \{ x' R x + u' Q u + \beta \, \mathbb{E} (A x + B u + C w_T)' P_T (A x + B u + C w_T) \} \tag{4.8}$$

To obtain the minimizer, we can take the derivative of the r.h.s. with respect to u and set it equal to zero.

Applying the relevant rules of matrix calculus, this gives

$$u = -(Q + \beta B' P_T B)^{-1} \beta B' P_T A x \tag{4.9}$$

Plugging this back into (4.8) and rearranging yields

$$J_{T-1}(x) = x' P_{T-1} x + d_{T-1}$$

where

$$P_{T-1} = R - \beta^2 A' P_T B (Q + \beta B' P_T B)^{-1} B' P_T A + \beta A' P_T A \tag{4.10}$$

and

$$d_{T-1} := \beta \operatorname{trace}(C' P_T C) \tag{4.11}$$

(The algebra is a good exercise — we'll leave it up to you.)

If we continue working backwards in this manner, it soon becomes clear that $J_t(x) = x' P_t x + d_t$ as claimed, where $\{P_t\}$ and $\{d_t\}$ satisfy the recursions

$$P_{t-1} = R - \beta^2 A' P_t B (Q + \beta B' P_t B)^{-1} B' P_t A + \beta A' P_t A \quad \text{with} \quad P_T = R_f \tag{4.12}$$

and

$$d_{t-1} = \beta (d_t + \operatorname{trace}(C' P_t C)) \quad \text{with} \quad d_T = 0 \tag{4.13}$$

Recalling (4.9), the minimizers from these backward steps are

$$u_t = -F_t x_t \quad \text{where} \quad F_t := (Q + \beta B' P_{t+1} B)^{-1} \beta B' P_{t+1} A \tag{4.14}$$

These are the linear optimal control policies we *discussed above*.

In particular, the sequence of controls given by (4.14) and (4.1) solves our finite horizon LQ problem.

Rephrasing this more precisely, the sequence $u_0, \dots, u_{T-1}$ given by

$$u_t = -F_t x_t \quad \text{with} \quad x_{t+1} = (A - BF_t) x_t + C w_{t+1} \tag{4.15}$$

for $t = 0, \dots, T-1$ attains the minimum of (4.6) subject to our constraints.

4.4 Implementation

We will use code from lqcontrol.py in QuantEcon.py to solve finite and infinite horizon linear quadratic control problems.

In the module, the various updating, simulation and fixed point methods are wrapped in a class called `LQ`, which includes

- Instance data:
  - The required parameters Q, R, A, B and optional parameters C, β, T, R_f, N specifying a given LQ model
    - set T and R_f to `None` in the infinite horizon case
    - set `C = None` (or zero) in the deterministic case
  - the value function and policy data
    - d_t, P_t, F_t in the finite horizon case

    * d, P, F in the infinite horizon case

- Methods:
    - `update_values` — shifts d_t, P_t, F_t to their $t-1$ values via (4.12), (4.13) and (4.14)
    - `stationary_values` — computes P, d, F in the infinite horizon case
    - `compute_sequence` — simulates the dynamics of x_t, u_t, w_t given x_0 and assuming standard normal shocks

4.4.1 An Application

Early Keynesian models assumed that households have a constant marginal propensity to consume from current income.

Data contradicted the constancy of the marginal propensity to consume.

In response, Milton Friedman, Franco Modigliani and others built models based on a consumer's preference for an intertemporally smooth consumption stream.

(See, for example, [Friedman, 1956] or [Modigliani and Brumberg, 1954].)

One property of those models is that households purchase and sell financial assets to make consumption streams smoother than income streams.

The household savings problem *outlined above* captures these ideas.

The optimization problem for the household is to choose a consumption sequence in order to minimize

$$\mathbb{E}\left\{\sum_{t=0}^{T-1}\beta^t(c_t-\bar{c})^2+\beta^T q a_T^2\right\} \tag{4.16}$$

subject to the sequence of budget constraints $a_{t+1}=(1+r)a_t-c_t+y_t,\ t\geq 0$.

Here q is a large positive constant, the role of which is to induce the consumer to target zero debt at the end of her life.

(Without such a constraint, the optimal choice is to choose $c_t=\bar{c}$ in each period, letting assets adjust accordingly.)

As before we set $y_t=\sigma w_{t+1}+\mu$ and $u_t:=c_t-\bar{c}$, after which the constraint can be written as in (4.2).

We saw how this constraint could be manipulated into the LQ formulation $x_{t+1}=Ax_t+Bu_t+Cw_{t+1}$ by setting $x_t=(a_t\ 1)'$ and using the definitions in (4.4).

To match with this state and control, the objective function (4.16) can be written in the form of (4.6) by choosing

$$Q:=1,\quad R:=\begin{pmatrix}0&0\\0&0\end{pmatrix},\quad\text{and}\quad R_f:=\begin{pmatrix}q&0\\0&0\end{pmatrix}$$

Now that the problem is expressed in LQ form, we can proceed to the solution by applying (4.12) and (4.14).

After generating shocks $w_1,\ldots,w_T$, the dynamics for assets and consumption can be simulated via (4.15).

The following figure was computed using $r=0.05,\beta=1/(1+r),\bar{c}=2,\mu=1,\sigma=0.25,T=45$ and $q=10^6$.

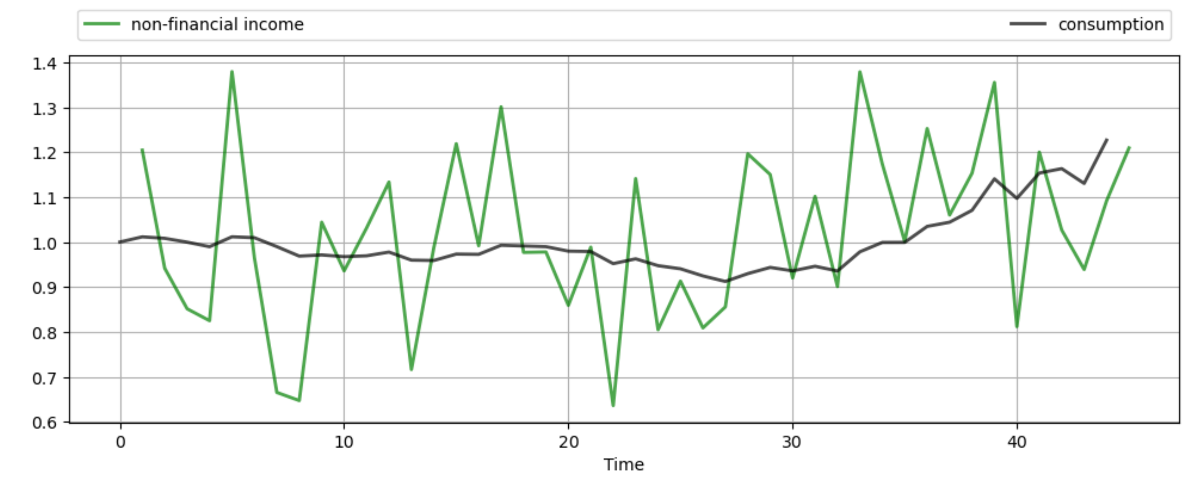
The shocks $\{w_t\}$ were taken to be IID and standard normal.

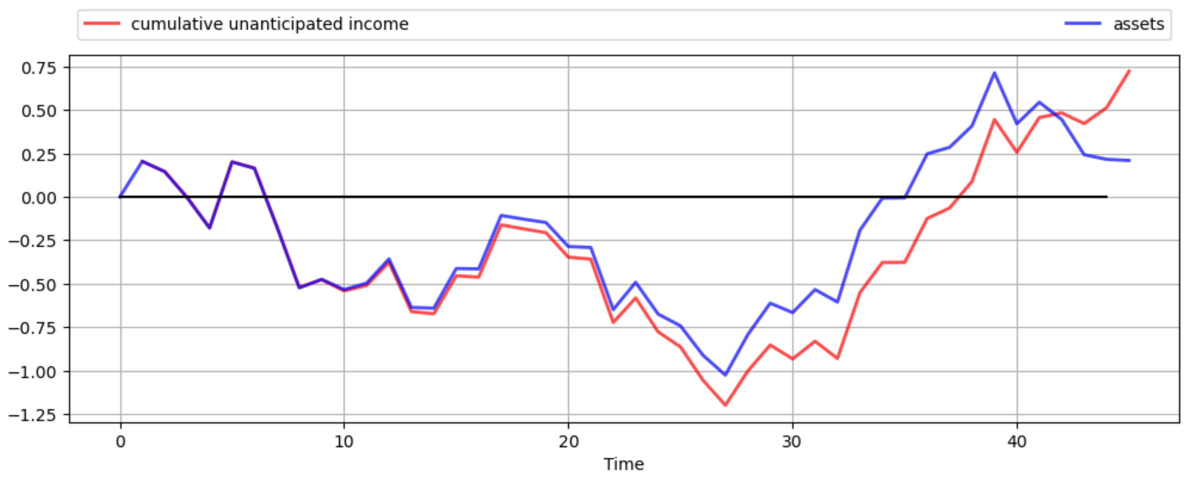
```
# Model parameters
r = 0.05
β = 1/(1 + r)
T = 45
c_bar = 2
σ = 0.25
```

(continues on next page)

(continued from previous page)

```
μ = 1
q = 1e6

# Formulate as an LQ problem
Q = 1
R = np.zeros((2, 2))
Rf = np.zeros((2, 2))
Rf[0, 0] = q
A = [[1 + r, -c_bar + μ],
     [0,              1]]
B = [[-1],
     [ 0]]
C = [[σ],
     [0]]

# Compute solutions and simulate
lq = LQ(Q, R, A, B, C, beta=β, T=T, Rf=Rf)
x0 = (0, 1)
xp, up, wp = lq.compute_sequence(x0)

# Convert back to assets, consumption and income
assets = xp[0, :]            # a_t
c = up.flatten() + c_bar     # c_t
income = σ * wp[0, 1:] + μ   # y_t

# Plot results
n_rows = 2
fig, axes = plt.subplots(n_rows, 1, figsize=(12, 10))

plt.subplots_adjust(hspace=0.5)

bbox = (0., 1.02, 1., .102)
legend_args = {'bbox_to_anchor': bbox, 'loc': 3, 'mode': 'expand'}
p_args = {'lw': 2, 'alpha': 0.7}

axes[0].plot(list(range(1, T+1)), income, 'g-', label="non-financial income",
             **p_args)
axes[0].plot(list(range(T)), c, 'k-', label="consumption", **p_args)

axes[1].plot(list(range(1, T+1)), np.cumsum(income - μ), 'r-',
             label="cumulative unanticipated income", **p_args)
axes[1].plot(list(range(T+1)), assets, 'b-', label="assets", **p_args)
axes[1].plot(list(range(T)), np.zeros(T), 'k-')

for ax in axes:
    ax.grid()
    ax.set_xlabel('Time')
    ax.legend(ncol=2, **legend_args)

plt.show()
```





The top panel shows the time path of consumption c_t and income y_t in the simulation.

As anticipated by the discussion on consumption smoothing, the time path of consumption is much smoother than that for income.

(But note that consumption becomes more irregular towards the end of life, when the zero final asset requirement impinges more on consumption choices.)

The second panel in the figure shows that the time path of assets a_t is closely correlated with cumulative unanticipated income, where the latter is defined as

$$z_t := \sum_{j=0}^{t} \sigma w_t$$

A key message is that unanticipated windfall gains are saved rather than consumed, while unanticipated negative shocks are met by reducing assets.

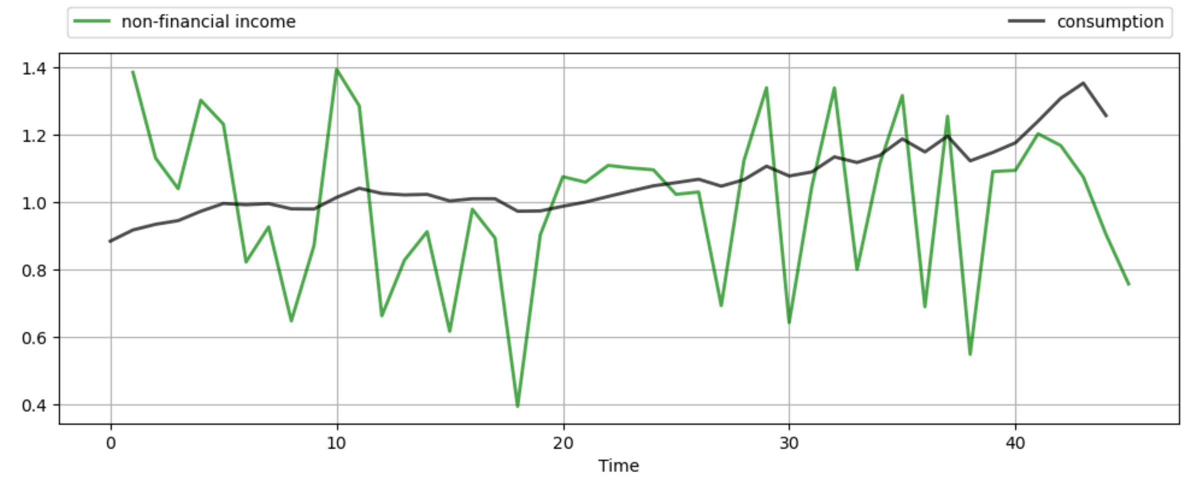
(Again, this relationship breaks down towards the end of life due to the zero final asset requirement.)

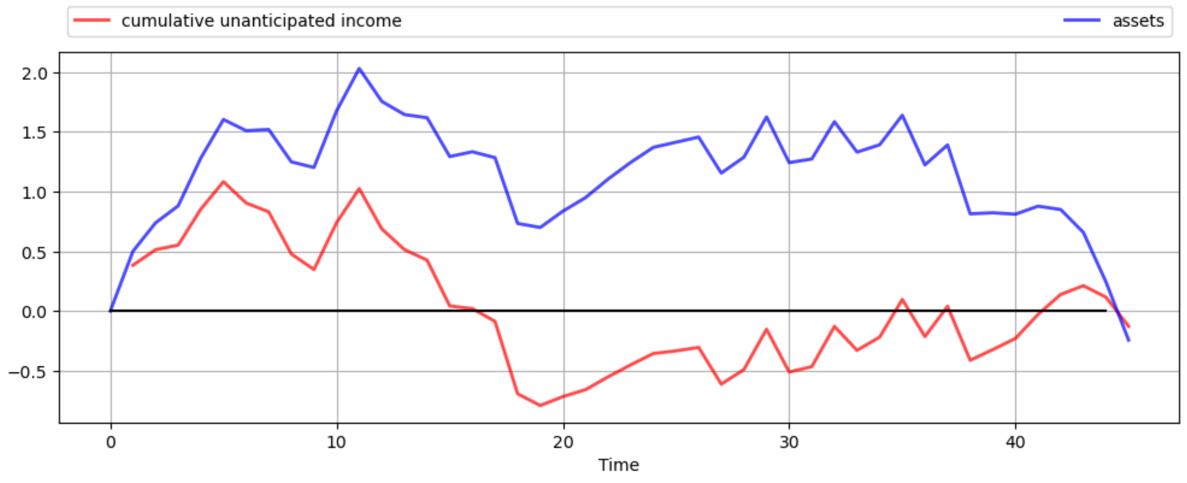
These results are relatively robust to changes in parameters.

For example, let's increase β from $1/(1+r) \approx 0.952$ to 0.96 while keeping other parameters fixed.

This consumer is slightly more patient than the last one, and hence puts relatively more weight on later consumption values.

```
# Compute solutions and simulate
lq = LQ(Q, R, A, B, C, beta=0.96, T=T, Rf=Rf)
x0 = (0, 1)
xp, up, wp = lq.compute_sequence(x0)

# Convert back to assets, consumption and income
assets = xp[0, :]            # a_t
c = up.flatten() + c_bar     # c_t
income = σ * wp[0, 1:] + μ   # y_t

# Plot results
n_rows = 2
fig, axes = plt.subplots(n_rows, 1, figsize=(12, 10))

plt.subplots_adjust(hspace=0.5)

bbox = (0., 1.02, 1., .102)
legend_args = {'bbox_to_anchor': bbox, 'loc': 3, 'mode': 'expand'}
p_args = {'lw': 2, 'alpha': 0.7}

axes[0].plot(list(range(1, T+1)), income, 'g-', label="non-financial income",
             **p_args)
axes[0].plot(list(range(T)), c, 'k-', label="consumption", **p_args)

axes[1].plot(list(range(1, T+1)), np.cumsum(income - μ), 'r-',
             label="cumulative unanticipated income", **p_args)
axes[1].plot(list(range(T+1)), assets, 'b-', label="assets", **p_args)
axes[1].plot(list(range(T)), np.zeros(T), 'k-')

for ax in axes:
    ax.grid()
    ax.set_xlabel('Time')
    ax.legend(ncol=2, **legend_args)

plt.show()
```





We now have a slowly rising consumption stream and a hump-shaped build-up of assets in the middle periods to fund rising consumption.

However, the essential features are the same: consumption is smooth relative to income, and assets are strongly positively correlated with cumulative unanticipated income.

4.5 Extensions and Comments

Let's now consider a number of standard extensions to the LQ problem treated above.

4.5.1 Time-Varying Parameters

In some settings, it can be desirable to allow A, B, C, R and Q to depend on t.

For the sake of simplicity, we've chosen not to treat this extension in our implementation given below.

However, the loss of generality is not as large as you might first imagine.

In fact, we can tackle many models with time-varying parameters by suitable choice of state variables.

One illustration is given *below*.

For further examples and a more systematic treatment, see [Hansen and Sargent, 2013], section 2.4.

4.5.2 Adding a Cross-Product Term

In some LQ problems, preferences include a cross-product term $u_t' N x_t$, so that the objective function becomes

$$\mathbb{E}\left\{\sum_{t=0}^{T-1}\beta^t(x_t'Rx_t + u_t'Qu_t + 2u_t'Nx_t) + \beta^T x_T' R_f x_T\right\} \tag{4.17}$$

Our results extend to this case in a straightforward way.

The sequence $\{P_t\}$ from (4.12) becomes

$$P_{t-1} = R - (\beta B' P_t A + N)'(Q + \beta B' P_t B)^{-1}(\beta B' P_t A + N) + \beta A' P_t A \quad \text{with} \quad P_T = R_f \tag{4.18}$$

The policies in (4.14) are modified to

$$u_t = -F_t x_t \quad \text{where} \quad F_t := (Q + \beta B' P_{t+1} B)^{-1}(\beta B' P_{t+1} A + N) \tag{4.19}$$

The sequence $\{d_t\}$ is unchanged from (4.13).

We leave interested readers to confirm these results (the calculations are long but not overly difficult).

4.5.3 Infinite Horizon

Finally, we consider the infinite horizon case, with *cross-product term*, unchanged dynamics and objective function given by

$$\mathbb{E}\left\{\sum_{t=0}^{\infty}\beta^t(x_t'Rx_t + u_t'Qu_t + 2u_t'Nx_t)\right\} \tag{4.20}$$

In the infinite horizon case, optimal policies can depend on time only if time itself is a component of the state vector x_t.

In other words, there exists a fixed matrix F such that $u_t = -Fx_t$ for all t.

That decision rules are constant over time is intuitive — after all, the decision-maker faces the same infinite horizon at every stage, with only the current state changing.

Not surprisingly, P and d are also constant.

The stationary matrix P is the solution to the discrete-time algebraic Riccati equation.

$$P = R - (\beta B' P A + N)'(Q + \beta B' P B)^{-1}(\beta B' P A + N) + \beta A' P A \tag{4.21}$$

Equation (4.21) is also called the *LQ Bellman equation*, and the map that sends a given P into the right-hand side of (4.21) is called the *LQ Bellman operator*.

The stationary optimal policy for this model is

$$u = -Fx \quad \text{where} \quad F = (Q + \beta B'PB)^{-1}(\beta B'PA + N) \tag{4.22}$$

The sequence $\{d_t\}$ from (4.13) is replaced by the constant value

$$d := \text{trace}(C'PC)\frac{\beta}{1-\beta} \tag{4.23}$$

The state evolves according to the time-homogeneous process $x_{t+1} = (A - BF)x_t + Cw_{t+1}$.

An example infinite horizon problem is treated *below*.

4.5.4 Certainty Equivalence

Linear quadratic control problems of the class discussed above have the property of *certainty equivalence*.

By this, we mean that the optimal policy F is not affected by the parameters in C, which specify the shock process.

This can be confirmed by inspecting (4.22) or (4.19).

It follows that we can ignore uncertainty when solving for optimal behavior, and plug it back in when examining optimal state dynamics.

4.6 Further Applications

4.6.1 Application 1: Age-Dependent Income Process

Previously we studied a permanent income model that generated consumption smoothing.

One unrealistic feature of that model is the assumption that the mean of the random income process does not depend on the consumer's age.

A more realistic income profile is one that rises in early working life, peaks towards the middle and maybe declines toward the end of working life and falls more during retirement.

In this section, we will model this rise and fall as a symmetric inverted "U" using a polynomial in age.

As before, the consumer seeks to minimize

$$\mathbb{E}\left\{\sum_{t=0}^{T-1}\beta^t(c_t - \bar{c})^2 + \beta^T q a_T^2\right\} \tag{4.24}$$

subject to $a_{t+1} = (1+r)a_t - c_t + y_t, \ t \geq 0$.

For income we now take $y_t = p(t) + \sigma w_{t+1}$ where $p(t) := m_0 + m_1 t + m_2 t^2$.

(In *the next section* we employ some tricks to implement a more sophisticated model.)

The coefficients m_0, m_1, m_2 are chosen such that $p(0) = 0, p(T/2) = \mu$, and $p(T) = 0$.

You can confirm that the specification $m_0 = 0, m_1 = T\mu/(T/2)^2, m_2 = -\mu/(T/2)^2$ satisfies these constraints.

To put this into an LQ setting, consider the budget constraint, which becomes

$$a_{t+1} = (1+r)a_t - u_t - \bar{c} + m_1 t + m_2 t^2 + \sigma w_{t+1} \tag{4.25}$$

The fact that a_{t+1} is a linear function of $(a_t, 1, t, t^2)$ suggests taking these four variables as the state vector x_t.

Once a good choice of state and control (recall $u_t = c_t - \bar{c}$) has been made, the remaining specifications fall into place relatively easily.

Thus, for the dynamics we set

$$x_t := \begin{pmatrix} a_t \\ 1 \\ t \\ t^2 \end{pmatrix}, \quad A := \begin{pmatrix} 1+r & -\bar{c} & m_1 & m_2 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 \end{pmatrix}, \quad B := \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad C := \begin{pmatrix} \sigma \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.26}$$

If you expand the expression $x_{t+1} = Ax_t + Bu_t + Cw_{t+1}$ using this specification, you will find that assets follow (4.25) as desired and that the other state variables also update appropriately.

To implement preference specification (4.24) we take

$$Q := 1, \quad R := \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad R_f := \begin{pmatrix} q & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.27}$$

The next figure shows a simulation of consumption and assets computed using the `compute_sequence` method of `lqcontrol.py` with initial assets set to zero.

Once again, smooth consumption is a dominant feature of the sample paths.

The asset path exhibits dynamics consistent with standard life cycle theory.

Exercise 4.7.1 gives the full set of parameters used here and asks you to replicate the figure.

4.6.2 Application 2: A Permanent Income Model with Retirement

In the *previous application*, we generated income dynamics with an inverted U shape using polynomials and placed them in an LQ framework.

It is arguably the case that this income process still contains unrealistic features.

A more common earning profile is where

1. income grows over working life, fluctuating around an increasing trend, with growth flattening off in later years
2. retirement follows, with lower but relatively stable (non-financial) income

Letting K be the retirement date, we can express these income dynamics by

$$y_t = \begin{cases} p(t) + \sigma w_{t+1} & \text{if } t \leq K \\ s & \text{otherwise} \end{cases} \tag{4.28}$$

Here

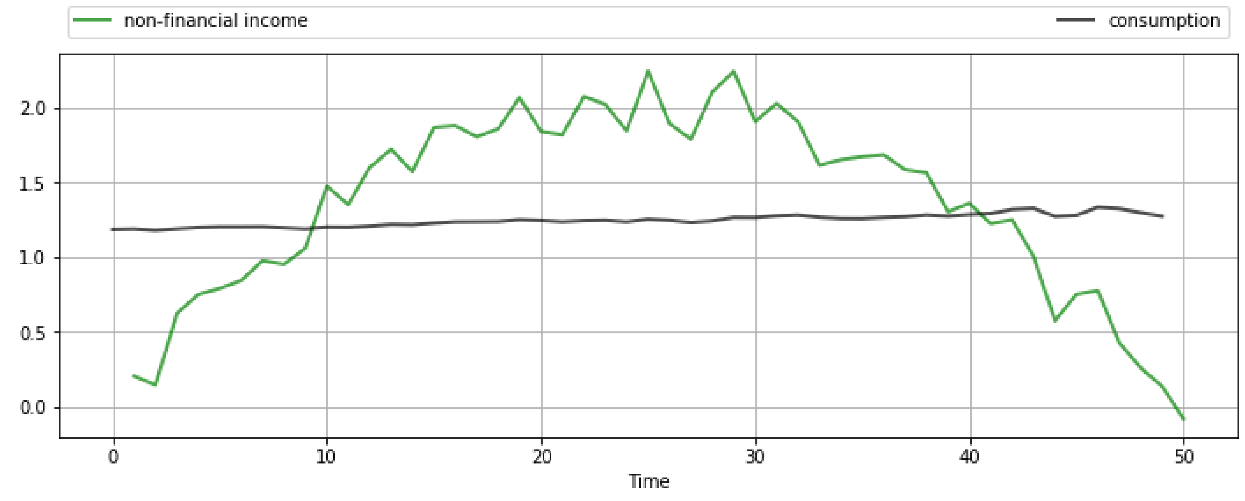
- $p(t) := m_1 t + m_2 t^2$ with the coefficients m_1, m_2 chosen such that $p(K) = \mu$ and $p(0) = p(2K) = 0$
- s is retirement income

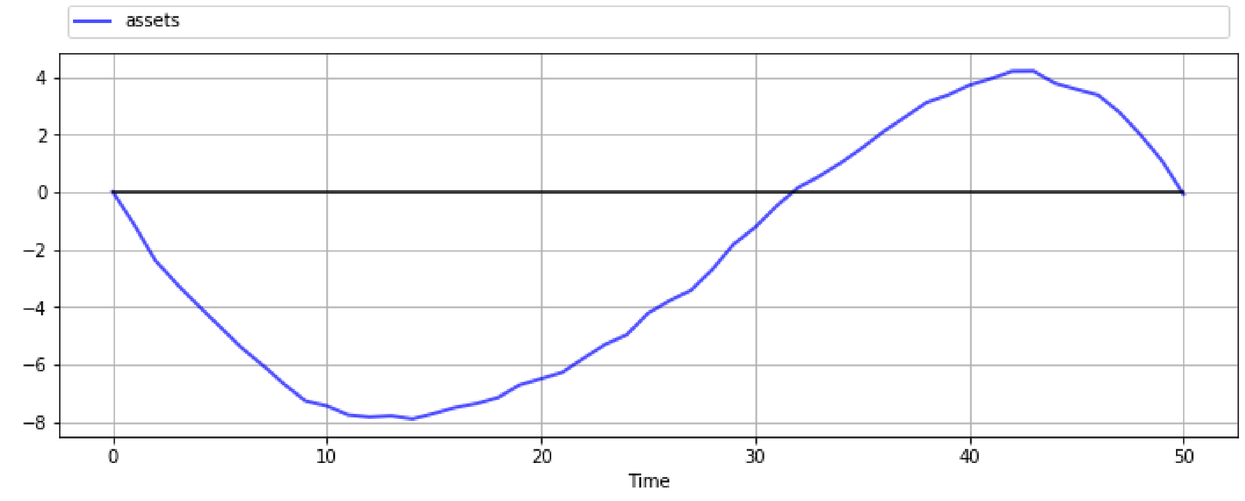
We suppose that preferences are unchanged and given by (4.16).

The budget constraint is also unchanged and given by $a_{t+1} = (1 + r)a_t - c_t + y_t$.

Our aim is to solve this problem and simulate paths using the LQ techniques described in this lecture.

In fact, this is a nontrivial problem, as the kink in the dynamics (4.28) at K makes it very difficult to express the law of motion as a fixed-coefficient linear system.


non-financial income
consumption
Time


assets
Time

However, we can still use our LQ methods here by suitably linking two-component LQ problems.

These two LQ problems describe the consumer's behavior during her working life (`lq_working`) and retirement (`lq_retired`).

(This is possible because, in the two separate periods of life, the respective income processes [polynomial trend and constant] each fit the LQ framework.)

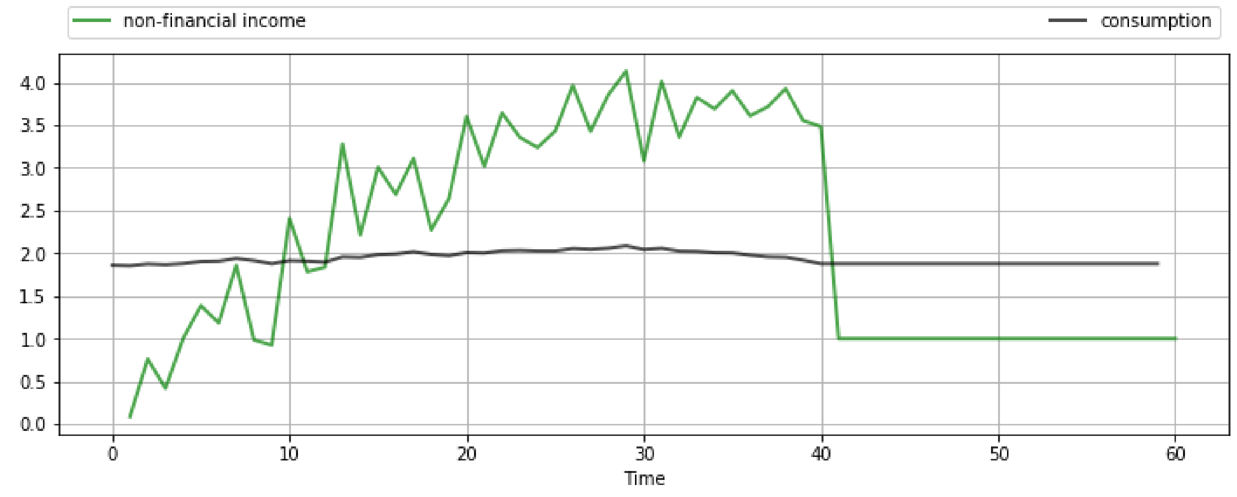
The basic idea is that although the whole problem is not a single time-invariant LQ problem, it is still a dynamic programming problem, and hence we can use appropriate Bellman equations at every stage.

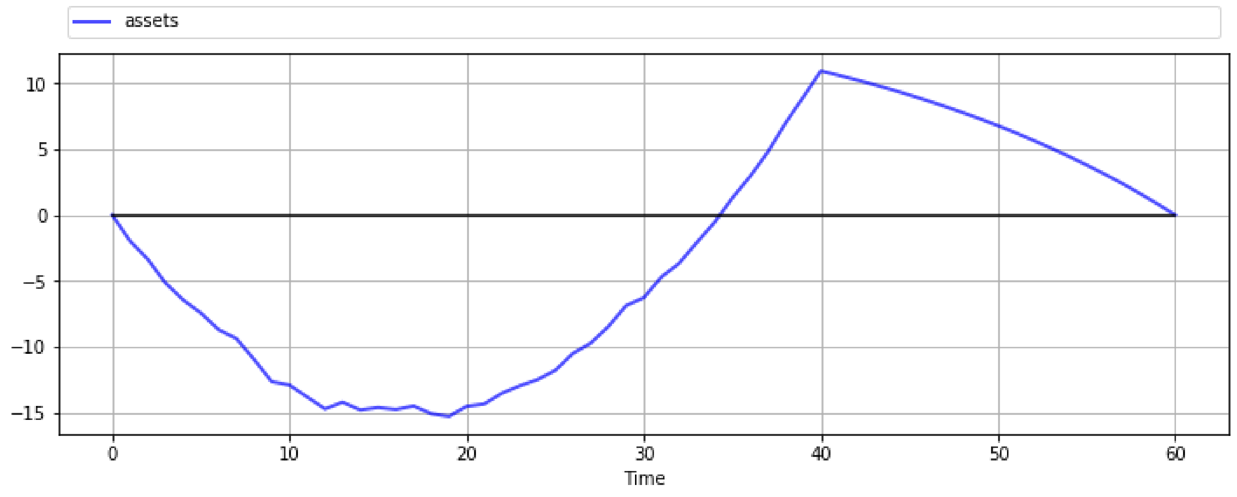
Based on this logic, we can

1. solve `lq_retired` by the usual backward induction procedure, iterating back to the start of retirement.
2. take the start-of-retirement value function generated by this process, and use it as the terminal condition R_f to feed into the `lq_working` specification.
3. solve `lq_working` by backward induction from this choice of R_f, iterating back to the start of working life.

This process gives the entire life-time sequence of value functions and optimal policies.

The next figure shows one simulation based on this procedure.





The full set of parameters used in the simulation is discussed in Exercise 4.7.2, where you are asked to replicate the figure.

Once again, the dominant feature observable in the simulation is consumption smoothing.

The asset path fits well with standard life cycle theory, with dissaving early in life followed by later saving.

Assets peak at retirement and subsequently decline.

4.6.3 Application 3: Monopoly with Adjustment Costs

Consider a monopolist facing stochastic inverse demand function

$$p_t = a_0 - a_1 q_t + d_t$$

Here q_t is output, and the demand shock d_t follows

$$d_{t+1} = \rho d_t + \sigma w_{t+1}$$

where $\{w_t\}$ is IID and standard normal.

The monopolist maximizes the expected discounted sum of present and future profits

$$\mathbb{E}\left\{\sum_{t=0}^{\infty} \beta^t \pi_t\right\} \quad \text{where} \quad \pi_t := p_t q_t - c q_t - \gamma(q_{t+1} - q_t)^2 \tag{4.29}$$

Here

- $\gamma(q_{t+1} - q_t)^2$ represents adjustment costs
- c is average cost of production

This can be formulated as an LQ problem and then solved and simulated, but first let's study the problem and try to get some intuition.

One way to start thinking about the problem is to consider what would happen if $\gamma = 0$.

Without adjustment costs there is no intertemporal trade-off, so the monopolist will choose output to maximize current profit in each period.

It's not difficult to show that profit-maximizing output is

$$\bar{q}_t := \frac{a_0 - c + d_t}{2a_1}$$

In light of this discussion, what we might expect for general γ is that

- if γ is close to zero, then q_t will track the time path of $\bar{q}_t$ relatively closely.
- if γ is larger, then q_t will be smoother than $\bar{q}_t$, as the monopolist seeks to avoid adjustment costs.

This intuition turns out to be correct.

The following figures show simulations produced by solving the corresponding LQ problem.

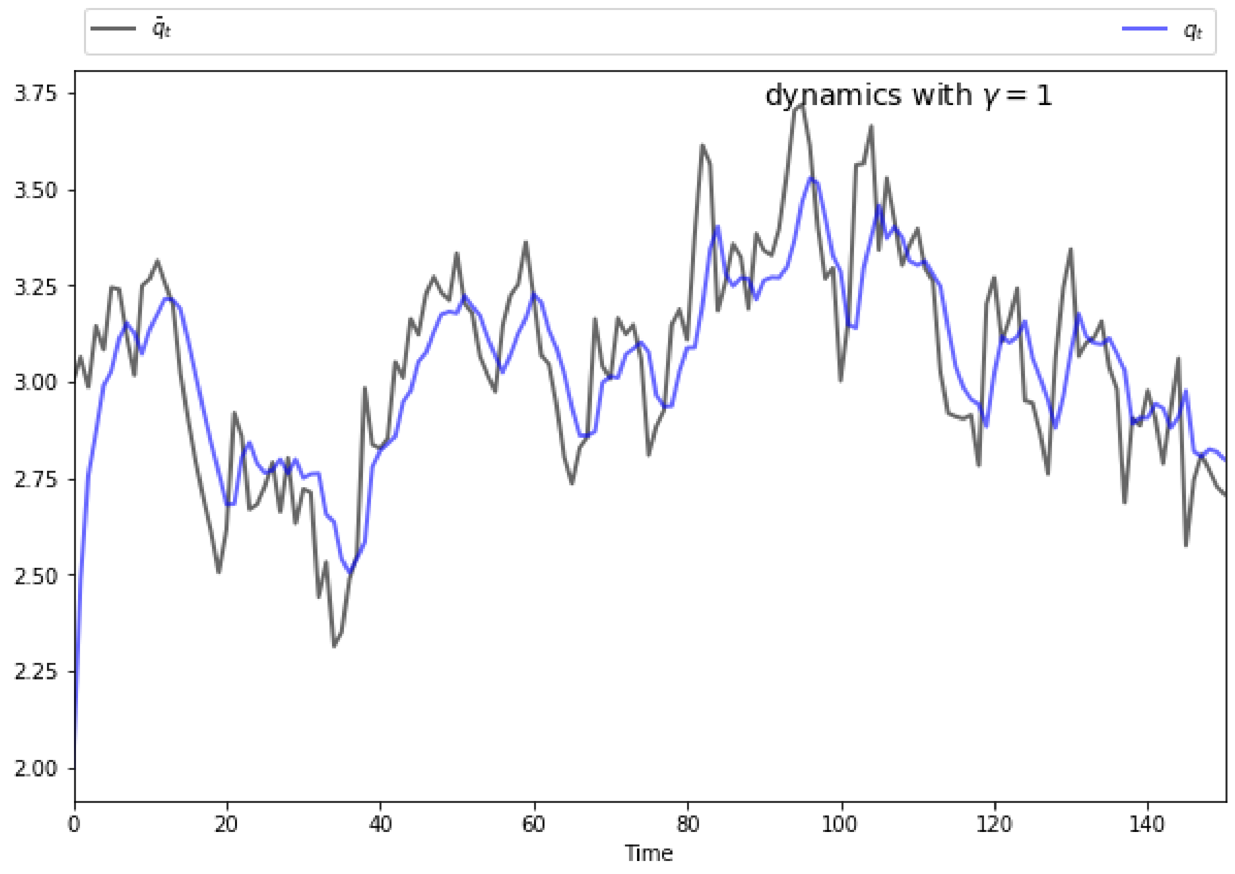
The only difference in parameters across the figures is the size of γ

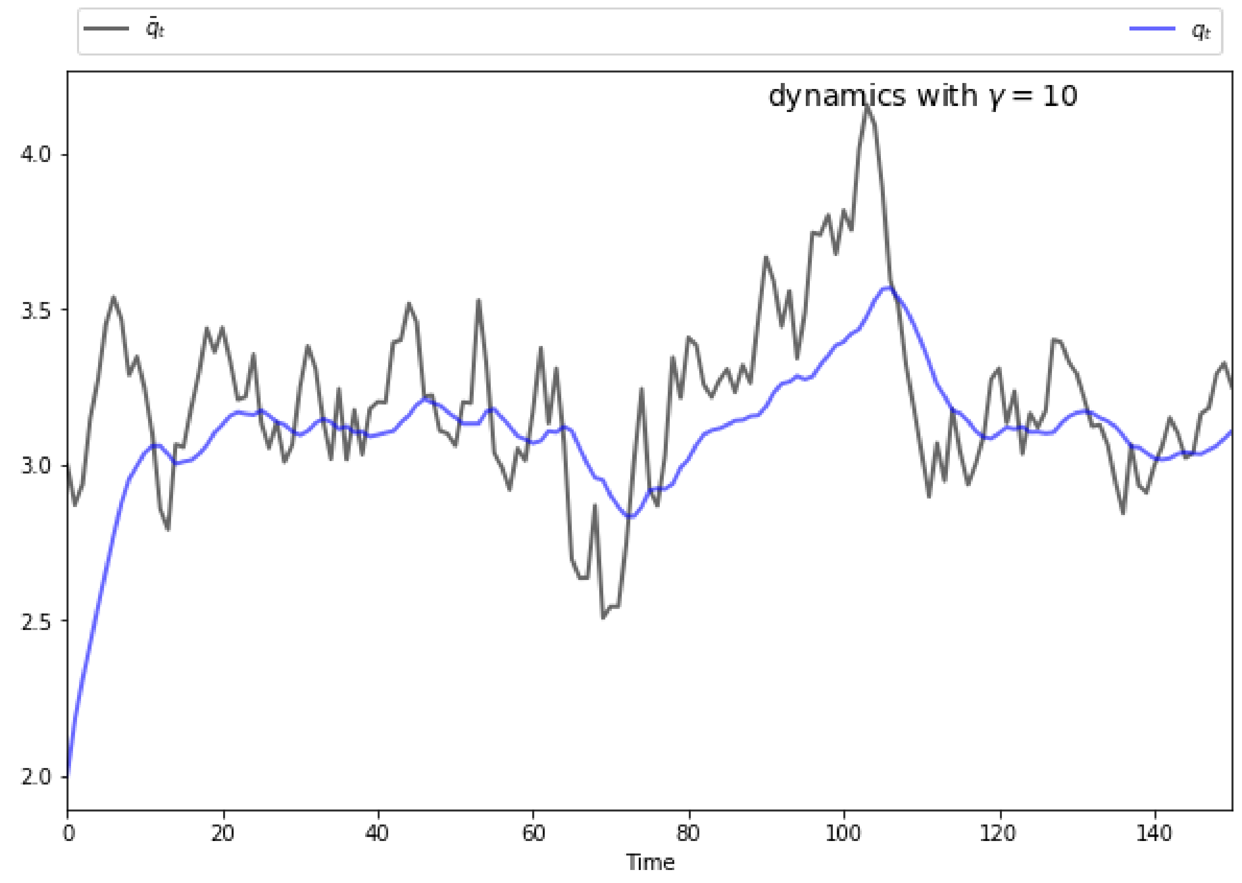
To produce these figures we converted the monopolist problem into an LQ problem.

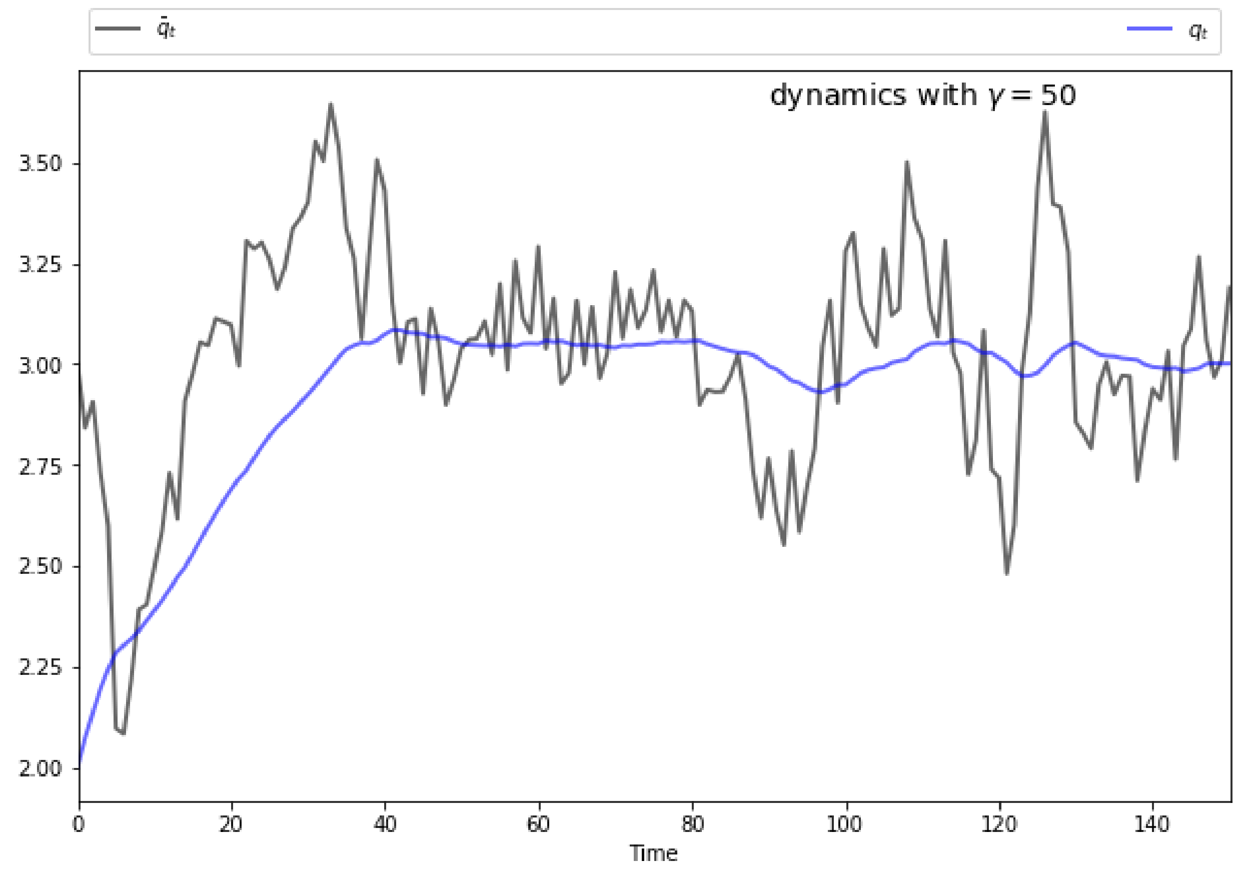
The key to this conversion is to choose the right state — which can be a bit of an art.

Here we take $x_t = (\bar{q}_t \;\; q_t \;\; 1)'$, while the control is chosen as $u_t = q_{t+1} - q_t$.

We also manipulated the profit function slightly.


$\bar{q}_t$
q_t
dynamics with $\gamma = 1$
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
0
20
40
60
80
100
120
140
Time


$\bar{q}_t$
q_t
dynamics with $\gamma = 10$
4.0
3.5
3.0
2.5
2.0
0
20
40
60
80
100
120
140
Time


$\bar{q}_t$
q_t
dynamics with $\gamma = 50$
3.50
3.25
3.00
2.75
2.50
2.25
2.00
0
20
40
60
80
100
120
140
Time

In (4.29), current profits are $\pi_t := p_t q_t - c q_t - \gamma(q_{t+1} - q_t)^2$.

Let's now replace π_t in (4.29) with $\hat{\pi}_t := \pi_t - a_1 \bar{q}_t^2$.

This makes no difference to the solution, since $a_1 \bar{q}_t^2$ does not depend on the controls.

(In fact, we are just adding a constant term to (4.29), and optimizers are not affected by constant terms.)

The reason for making this substitution is that, as you will be able to verify, $\hat{\pi}_t$ reduces to the simple quadratic

$$\hat{\pi}_t = -a_1(q_t - \bar{q}_t)^2 - \gamma u_t^2$$

After negation to convert to a minimization problem, the objective becomes

$$\min \mathbb{E} \sum_{t=0}^{\infty} \beta^t \left\{ a_1(q_t - \bar{q}_t)^2 + \gamma u_t^2 \right\} \tag{4.30}$$

It's now relatively straightforward to find R and Q such that (4.30) can be written as (4.20).

Furthermore, the matrices A, B and C from (4.1) can be found by writing down the dynamics of each element of the state.

Exercise 4.7.3 asks you to complete this process, and reproduce the preceding figures.

4.7 Exercises

Exercise 4.7.1

Replicate the figure with polynomial income *shown above*.

The parameters are $r = 0.05, \beta = 1/(1+r), \bar{c} = 1.5, \mu = 2, \sigma = 0.15, T = 50$ and $q = 10^4$.

Solution to Exercise 4.7.1

Here's one solution.

We use some fancy plot commands to get a certain style — feel free to use simpler ones.

The model is an LQ permanent income / life-cycle model with hump-shaped income

$$y_t = m_1 t + m_2 t^2 + \sigma w_{t+1}$$

where $\{w_t\}$ is IID $N(0,1)$ and the coefficients m_1 and m_2 are chosen so that $p(t) = m_1 t + m_2 t^2$ has an inverted U shape with

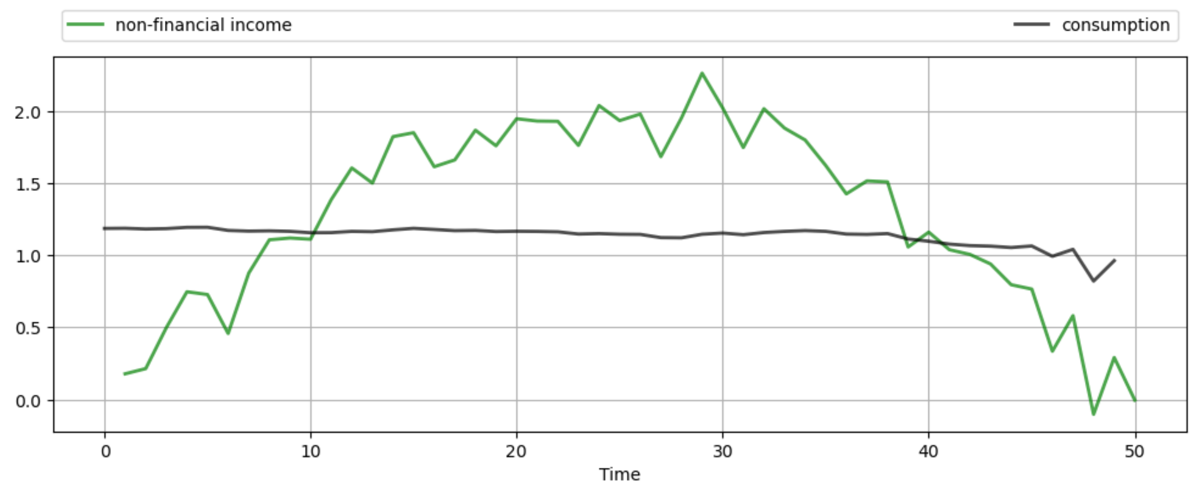
- $p(0) = 0, p(T/2) = \mu$, and
- $p(T) = 0$

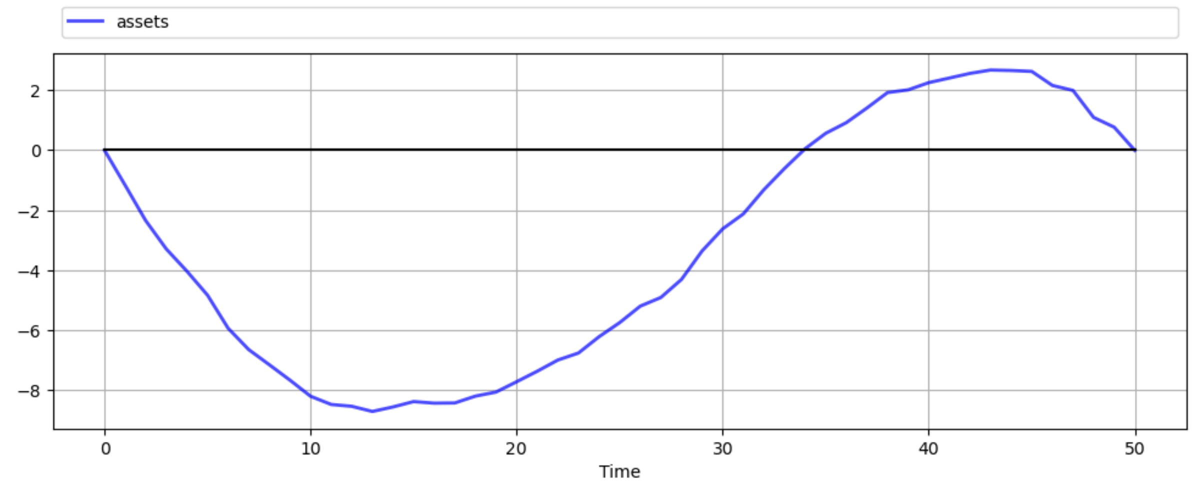
```
# Model parameters
r = 0.05
β = 1/(1 + r)
T = 50
c_bar = 1.5
σ = 0.15
μ = 2
```

(continues on next page)

(continued from previous page)

```
q = 1e4
m1 = T * (μ/(T/2)**2)
m2 = -(μ/(T/2)**2)

# Formulate as an LQ problem
Q = 1
R = np.zeros((4, 4))
Rf = np.zeros((4, 4))
Rf[0, 0] = q
A = [[1 + r, -c_bar, m1, m2],
     [0,          1,  0,  0],
     [0,          1,  1,  0],
     [0,          1,  2,  1]]
B = [[-1],
     [ 0],
     [ 0],
     [ 0]]
C = [[σ],
     [0],
     [0],
     [0]]

# Compute solutions and simulate
lq = LQ(Q, R, A, B, C, beta=β, T=T, Rf=Rf)
x0 = (0, 1, 0, 0)
xp, up, wp = lq.compute_sequence(x0)

# Convert results back to assets, consumption and income
ap = xp[0, :]               # Assets
c = up.flatten() + c_bar    # Consumption
time = np.arange(1, T+1)
income = σ * wp[0, 1:] + m1 * time + m2 * time**2  # Income


# Plot results
n_rows = 2
fig, axes = plt.subplots(n_rows, 1, figsize=(12, 10))

plt.subplots_adjust(hspace=0.5)

bbox = (0., 1.02, 1., .102)
legend_args = {'bbox_to_anchor': bbox, 'loc': 3, 'mode': 'expand'}
p_args = {'lw': 2, 'alpha': 0.7}

axes[0].plot(range(1, T+1), income, 'g-', label="non-financial income",
            **p_args)
axes[0].plot(range(T), c, 'k-', label="consumption", **p_args)

axes[1].plot(range(T+1), ap.flatten(), 'b-', label="assets", **p_args)
axes[1].plot(range(T+1), np.zeros(T+1), 'k-')

for ax in axes:
    ax.grid()
    ax.set_xlabel('Time')
    ax.legend(ncol=2, **legend_args)

plt.show()
```





Exercise 4.7.2

Replicate the figure on work and retirement *shown above*.

The parameters are $r = 0.05, \beta = 1/(1+r), \bar{c} = 4, \mu = 4, \sigma = 0.35, K = 40, T = 60, s = 1$ and $q = 10^4$.

To understand the overall procedure, carefully read the section containing that figure.

Hint: First, in order to make our approach work, we must ensure that both LQ problems have the same state variables and control.

As with previous applications, the control can be set to $u_t = c_t - \bar{c}$.

For `lq_working`, x_t, A, B, C can be chosen as in (4.26).

- Recall that m_1, m_2 are chosen so that $p(K) = \mu$ and $p(2K) = 0$.

For `lq_retired`, use the same definition of x_t and u_t, but modify A, B, C to correspond to constant income $y_t = s$.

For `lq_retired`, set preferences as in (4.27).

For `lq_working`, preferences are the same, except that R_f should be replaced by the final value function that emerges from iterating `lq_retired` back to the start of retirement.

With some careful footwork, the simulation can be generated by patching together the simulations from these two separate models.

Solution to Exercise 4.7.2

This is a permanent income / life-cycle model with polynomial growth in income over working life followed by a fixed retirement income.

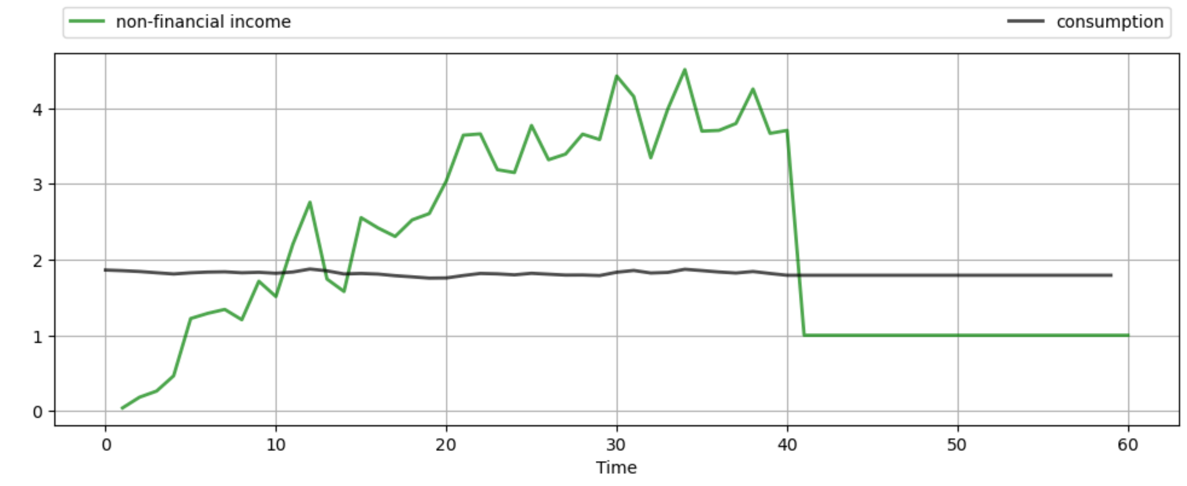
The model is solved by combining two LQ programming problems as described in the lecture.

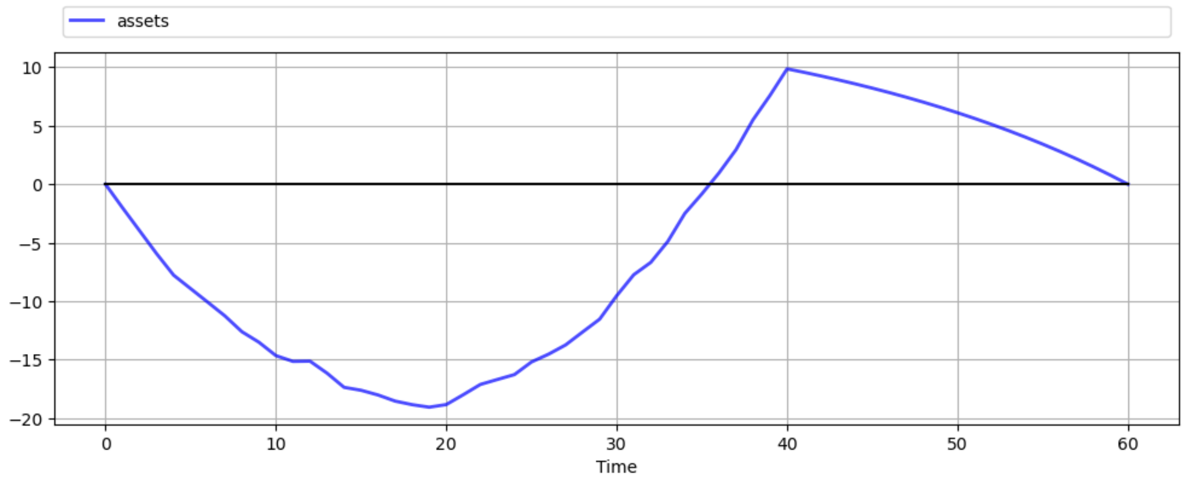
```
# Model parameters
r = 0.05
β = 1/(1 + r)
T = 60
K = 40
c_bar = 4
σ = 0.35
μ = 4
q = 1e4
s = 1
m1 = 2 * μ/K
m2 = -μ/K**2

# Formulate LQ problem 1 (retirement)
Q = 1
R = np.zeros((4, 4))
Rf = np.zeros((4, 4))
Rf[0, 0] = q
A = [[1 + r, s - c_bar, 0, 0],
     [0,              1, 0, 0],
     [0,              1, 1, 0],
     [0,              1, 2, 1]]
B = [[-1],
     [ 0],
     [ 0],
     [ 0]]
C = [[0],
     [0],
     [0],
     [0]]

# Initialize LQ instance for retired agent
lq_retired = LQ(Q, R, A, B, C, beta=β, T=T-K, Rf=Rf)
# Iterate back to start of retirement, record final value function
for i in range(T-K):
    lq_retired.update_values()
Rf2 = lq_retired.P

# Formulate LQ problem 2 (working life)
R = np.zeros((4, 4))
A = [[1 + r, -c_bar, m1, m2],
     [0,          1,  0,  0],
     [0,          1,  1,  0],
```

(continues on next page)

(continued from previous page)

```
     [0,          1,  2,  1]]
B = [[-1],
     [ 0],
     [ 0],
     [ 0]]
C = [[σ],
     [0],
     [0],
     [0]]

# Set up working life LQ instance with terminal Rf from lq_retired
lq_working = LQ(Q, R, A, B, C, beta=β, T=K, Rf=Rf2)

# Simulate working state / control paths
x0 = (0, 1, 0, 0)
xp_w, up_w, wp_w = lq_working.compute_sequence(x0)
# Simulate retirement paths (note the initial condition)
xp_r, up_r, wp_r = lq_retired.compute_sequence(xp_w[:, K])

# Convert results back to assets, consumption and income
xp = np.column_stack((xp_w, xp_r[:, 1:]))
assets = xp[0, :]                  # Assets

up = np.column_stack((up_w, up_r))
c = up.flatten() + c_bar           # Consumption

time = np.arange(1, K+1)
income_w = σ * wp_w[0, 1:K+1] + m1 * time + m2 * time**2  # Income
income_r = np.full(T-K, s)
income = np.concatenate((income_w, income_r))

# Plot results
n_rows = 2
fig, axes = plt.subplots(n_rows, 1, figsize=(12, 10))

plt.subplots_adjust(hspace=0.5)

bbox = (0., 1.02, 1., .102)
legend_args = {'bbox_to_anchor': bbox, 'loc': 3, 'mode': 'expand'}
p_args = {'lw': 2, 'alpha': 0.7}

axes[0].plot(range(1, T+1), income, 'g-', label="non-financial income",
            **p_args)
axes[0].plot(range(T), c, 'k-', label="consumption", **p_args)

axes[1].plot(range(T+1), assets, 'b-', label="assets", **p_args)
axes[1].plot(range(T+1), np.zeros(T+1), 'k-')

for ax in axes:
    ax.grid()
    ax.set_xlabel('Time')
    ax.legend(ncol=2, **legend_args)

plt.show()
```





Exercise 4.7.3

Reproduce the figures from the monopolist application *given above*.

For parameters, use $a_0 = 5, a_1 = 0.5, \sigma = 0.15, \rho = 0.9, \beta = 0.95$ and $c = 2$, while γ varies between 1 and 50 (see figures).

Solution to Exercise 4.7.3

The first task is to find the matrices A, B, C, Q, R that define the LQ problem.

Recall that $x_t = (\bar{q}_t \;\; q_t \;\; 1)'$, while $u_t = q_{t+1} - q_t$.

Letting $m_0 := (a_0 - c)/2a_1$ and $m_1 := 1/2a_1$, we can write $\bar{q}_t = m_0 + m_1 d_t$, and then, with some manipulation

$$\bar{q}_{t+1} = m_0(1 - \rho) + \rho \bar{q}_t + m_1 \sigma w_{t+1}$$

By our definition of u_t, the dynamics of q_t are $q_{t+1} = q_t + u_t$.

Using these facts you should be able to build the correct A, B, C matrices (and then check them against those found in the solution code below).

Suitable R, Q matrices can be found by inspecting the objective function, which we repeat here for convenience:

$$\min \mathbb{E} \left\{ \sum_{t=0}^{\infty} \beta^t a_1 (q_t - \bar{q}_t)^2 + \gamma u_t^2 \right\}$$

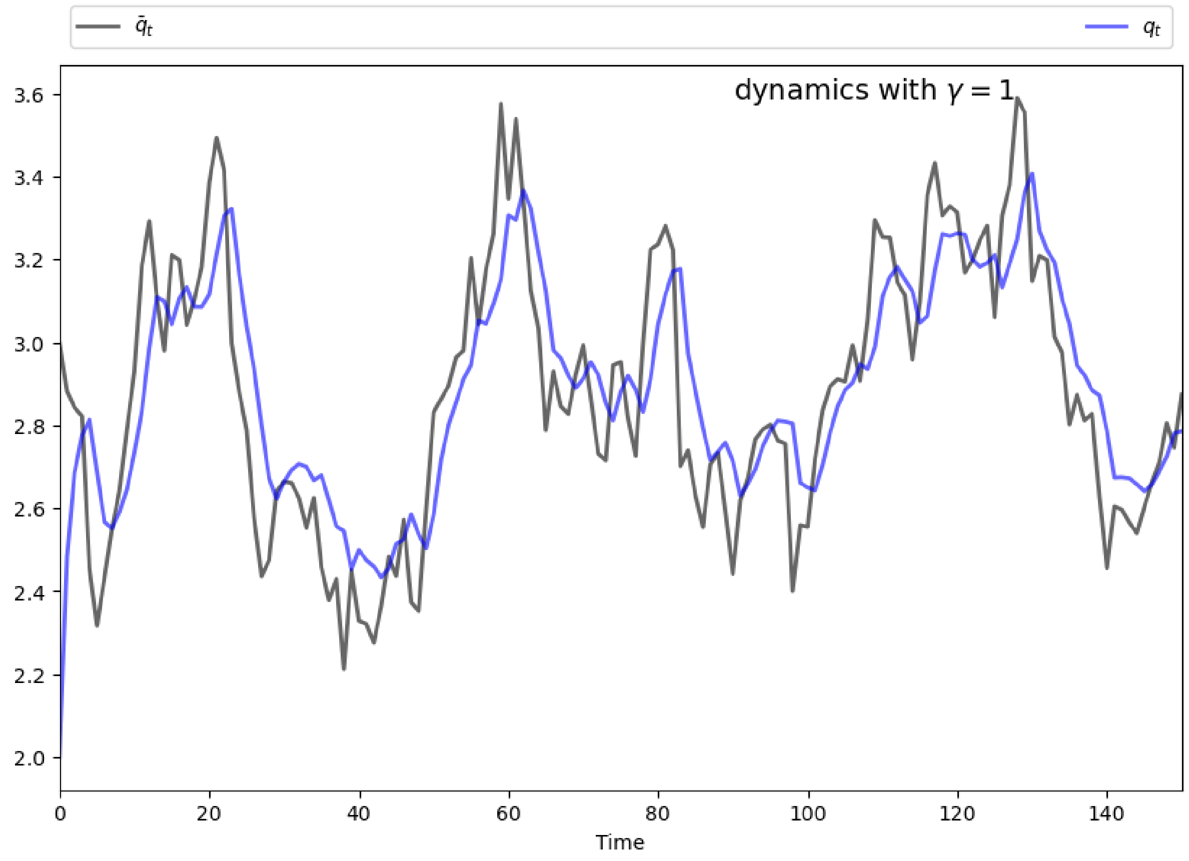
Our solution code is

```
# Model parameters
a0 = 5
a1 = 0.5
σ = 0.15
ρ = 0.9
γ = 1
β = 0.95
c = 2
T = 120

# Useful constants
m0 = (a0-c)/(2 * a1)
m1 = 1/(2 * a1)

# Formulate LQ problem
Q = γ
R = [[ a1, -a1,  0],
     [-a1,  a1,  0],
     [  0,   0,  0]]
A = [[ρ, 0, m0 * (1 - ρ)],
     [0, 1,            0],
     [0, 0,            1]]

B = [[0],
     [1],
     [0]]
C = [[m1 * σ],
     [     0],
     [     0]]

lq = LQ(Q, R, A, B, C=C, beta=β)

# Simulate state / control paths
x0 = (m0, 2, 1)
xp, up, wp = lq.compute_sequence(x0, ts_length=150)
q_bar = xp[0, :]
q = xp[1, :]

# Plot simulation results
fig, ax = plt.subplots(figsize=(10, 6.5))

# Some fancy plotting stuff -- simplify if you prefer
bbox = (0., 1.01, 1., .101)
legend_args = {'bbox_to_anchor': bbox, 'loc': 3, 'mode': 'expand'}
p_args = {'lw': 2, 'alpha': 0.6}

time = range(len(q))
ax.set(xlabel='Time', xlim=(0, max(time)))
ax.plot(time, q_bar, 'k-', lw=2, alpha=0.6, label=r'$\bar q_t$')
ax.plot(time, q, 'b-', lw=2, alpha=0.6, label='$q_t$')
ax.legend(ncol=2, **legend_args)
```

(continues on next page)

(continued from previous page)

```
s = f'dynamics with $\gamma = {γ}$'
ax.text(max(time) * 0.6, 1 * q_bar.max(), s, fontsize=14)
plt.show()
```



CHAPTER

FIVE

LAGRANGIAN FOR LQ CONTROL

```
!pip install quantecon
```

```
import numpy as np
from quantecon import LQ
from scipy.linalg import schur
```

5.1 Overview

This is a sequel to this lecture *linear quadratic dynamic programming*

It can also be regarded as presenting **invariant subspace** techniques that extend ones that we encountered earlier in this lecture stability in linear rational expectations models

We present a Lagrangian formulation of an infinite horizon linear quadratic undiscounted dynamic programming problem.

Such a problem is also sometimes called an optimal linear regulator problem.

A Lagrangian formulation

- carries insights about connections between stability and optimality
- is the basis for fast algorithms for solving Riccati equations
- opens the way to constructing solutions of dynamic systems that don't come directly from an intertemporal optimization problem

A key tool in this lecture is the concept of an $n \times n$ **symplectic** matrix.

A symplectic matrix has eigenvalues that occur in **reciprocal pairs**, meaning that if $\lambda_i \in (-1, 1)$ is an eigenvalue, then so is λ_i^{-1}.

This reciprocal pairs property of the eigenvalues of a matrix is a tell-tale sign that the matrix describes the joint dynamics of a system of equations describing the **states** and **costates** that constitute first-order necessary conditions for solving an undiscounted linear-quadratic infinite-horizon optimization problem.

The symplectic matrix that will interest us describes the first-order dynamics of **state** and **co-state** vectors of an optimally controlled system.

In focusing on eigenvalues and eigenvectors of this matrix, we capitalize on an analysis of **invariant subspaces.**

These invariant subspace formulations of LQ dynamic programming problems provide a bridge between recursive (i.e., dynamic programming) formulations and classical formulations of linear control and linear filtering problems that make use of related matrix decompositions (see for example this lecture and this lecture).

While most of this lecture focuses on undiscounted problems, later sections describe handy ways of transforming discounted problems to undiscounted ones.

The techniques in this lecture will prove useful when we study Stackelberg and Ramsey problem in this lecture.

5.2 Undiscounted LQ DP Problem

The problem is to choose a sequence of controls $\{u_t\}_{t=0}^\infty$ to maximize the criterion

$$-\sum_{t=0}^{\infty}\{x_t' R x_t + u_t' Q u_t\}$$

subject to $x_{t+1} = Ax_t + Bu_t$, where x_0 is a given initial state vector.

Here x_t is an $(n \times 1)$ vector of state variables, u_t is a $(k \times 1)$ vector of controls, R is a positive semidefinite symmetric matrix, Q is a positive definite symmetric matrix, A is an $(n \times n)$ matrix, and B is an $(n \times k)$ matrix.

The optimal value function turns out to be quadratic, $V(x) = -x'Px$, where P is a positive semidefinite symmetric matrix.

Using the transition law to eliminate next period's state, the Bellman equation becomes

$$-x'Px = \max_u\{-x'Rx - u'Qu - (Ax + Bu)'P(Ax + Bu)\} \tag{5.1}$$

The first-order necessary conditions for the maximum problem on the right side of equation (5.1) are

Note: We use the following rules for differentiating quadratic and bilinear matrix forms: $\frac{\partial x' A x}{\partial x} = (A + A')x$; $\frac{\partial y' B z}{\partial y} = Bz$, $\frac{\partial y' B z}{\partial z} = B'y$.

$$(Q + B'PB)u = -B'PAx,$$

which implies that an optimal decision rule for u is

$$u = -(Q + B'PB)^{-1}B'PAx$$

or

$$u = -Fx,$$

where

$$F = (Q + B'PB)^{-1}B'PA.$$

Substituting $u = -(Q + B'PB)^{-1}B'PAx$ into the right side of equation (5.1) and rearranging gives

$$P = R + A'PA - A'PB(Q + B'PB)^{-1}B'PA. \tag{5.2}$$

Equation (5.2) is called an **algebraic matrix Riccati** equation.

There are multiple solutions of equation (5.2).

But only one of them is positive definite.

The positive define solution is associated with the maximum of our problem.

It expresses the matrix P as an implicit function of the matrices R, Q, A, B.

Notice that the **gradient of the value function** is

$$\frac{\partial V(x)}{\partial x} = -2Px \tag{5.3}$$

We shall use fact (5.3) later.

5.3 Lagrangian

For the undiscounted optimal linear regulator problem, form the Lagrangian

$$L = -\sum_{t=0}^{\infty} \biggl\{ x_t' R x_t + u_t' Q u_t + 2\mu_{t+1}' [A x_t + B u_t - x_{t+1}] \biggr\} \tag{5.4}$$

where $2\mu_{t+1}$ is a vector of Lagrange multipliers on the time t transition law $x_{t+1} = Ax_t + Bu_t$.

(We put the 2 in front of μ_{t+1} to make things match up nicely with equation (5.3).)

First-order conditions for maximization with respect to $\{u_t, x_{t+1}\}_{t=0}^{\infty}$ are

$$\begin{aligned} 2Qu_t + 2B'\mu_{t+1} &= 0 \ , \ t \geq 0 \\ \mu_t = Rx_t + A'\mu_{t+1} &\ , \ t \geq 1. \end{aligned} \tag{5.5}$$

Define μ_0 to be a vector of shadow prices of x_0 and apply an envelope condition to (5.4) to deduce that

$$\mu_0 = Rx_0 + A'\mu_1,$$

which is a time $t = 0$ counterpart to the second equation of system (5.5).

An important fact is that

$$\mu_{t+1} = Px_{t+1} \tag{5.6}$$

where P is a positive define matrix that solves the algebraic Riccati equation (5.2).

Thus, from equations (5.3) and (5.6), $-2\mu_t$ is the gradient of the value function with respect to x_t.

The Lagrange multiplier vector μ_t is often called the **costate** vector that corresponds to the **state** vector x_t.

It is useful to proceed with the following steps:

- solve the first equation of (5.5) for u_t in terms of μ_{t+1}.
- substitute the result into the law of motion $x_{t+1} = Ax_t + Bu_t$.
- arrange the resulting equation and the second equation of (5.5) into the form

$$L \begin{pmatrix} x_{t+1} \\ \mu_{t+1} \end{pmatrix} = N \begin{pmatrix} x_t \\ \mu_t \end{pmatrix} \ , \ t \geq 0, \tag{5.7}$$

where

$$L = \begin{pmatrix} I & BQ^{-1}B' \\ 0 & A' \end{pmatrix}, \quad N = \begin{pmatrix} A & 0 \\ -R & I \end{pmatrix}.$$

When L is of full rank (i.e., when A is of full rank), we can write system (5.7) as

$$\begin{pmatrix} x_{t+1} \\ \mu_{t+1} \end{pmatrix} = M \begin{pmatrix} x_t \\ \mu_t \end{pmatrix} \tag{5.8}$$

where

$$M \equiv L^{-1} N = \begin{pmatrix} A + BQ^{-1}B'A'^{-1}R & -BQ^{-1}B'A'^{-1} \\ -A'^{-1}R & A'^{-1} \end{pmatrix}. \tag{5.9}$$

5.4 State-Costate Dynamics

We seek to solve the difference equation system (5.8) for a sequence $\{x_t\}_{t=0}^{\infty}$ that satisfies

- an initial condition for x_0
- a terminal condition $\lim_{t \rightarrow +\infty} x_t = 0$

This terminal condition reflects our desire for a **stable** solution, one that does not diverge as $t \rightarrow \infty$.

We inherit our wish for stability of the $\{x_t\}$ sequence from a desire to maximize

$$-\sum_{t=0}^{\infty}[x_t'Rx_t + u_t'Qu_t],$$

which requires that $x_t'Rx_t$ converge to zero as $t \rightarrow +\infty$.

5.5 Reciprocal Pairs Property

To proceed, we study properties of the $(2n \times 2n)$ matrix M defined in (5.9).

It helps to introduce a $(2n \times 2n)$ matrix

$$J = \begin{pmatrix} 0 & -I_n \\ I_n & 0 \end{pmatrix}.$$

The rank of J is $2n$.

Definition: A matrix M is called **symplectic** if

$$MJM' = J. \tag{5.10}$$

Salient properties of symplectic matrices that are readily verified include:

- If M is symplectic, then M^2 is symplectic
- The determinant of a symplectic, then $\det(M) = 1$

It can be verified directly that M in equation (5.9) is symplectic.

It follows from equation (5.10) and from the fact $J^{-1} = J' = -J$ that for any symplectic matrix M,

$$M' = J^{-1}M^{-1}J. \tag{5.11}$$

Equation (5.11) states that M' is related to the inverse of M by a **similarity transformation**.

For square matrices, recall that

- similar matrices share eigenvalues
- eigenvalues of the inverse of a matrix are inverses of eigenvalues of the matrix
- a matrix and its transpose share eigenvalues

It then follows from equation (5.11) that the eigenvalues of M occur in reciprocal pairs: if λ is an eigenvalue of M, so is λ^{-1}.

Write equation (5.8) as

$$y_{t+1} = My_t \tag{5.12}$$

where $y_t = \begin{pmatrix} x_t \\ \mu_t \end{pmatrix}$.

Consider a **triangularization** of M

$$V^{-1}MV = \begin{pmatrix} W_{11} & W_{12} \\ 0 & W_{22} \end{pmatrix} \tag{5.13}$$

where

- each block on the right side is $(n \times n)$
- V is nonsingular
- all eigenvalues of W_{22} exceed 1 in modulus
- all eigenvalues of W_{11} are less than 1 in modulus

5.6 Schur decomposition

The **Schur decomposition** and the **eigenvalue decomposition** are two decompositions of the form (5.13).

Write equation (5.12) as

$$y_{t+1} = VWV^{-1}y_t. \tag{5.14}$$

A solution of equation (5.14) for arbitrary initial condition y_0 is evidently

$$y_t = V \begin{bmatrix} W_{11}^t & W_{12,t} \\ 0 & W_{22}^t \end{bmatrix} V^{-1}y_0 \tag{5.15}$$

where $W_{12,t} = W_{12}$ for $t = 1$ and for $t \geq 2$ obeys the recursion

$$W_{12,t} = W_{11}^{t-1}W_{12,t-1} + W_{12,t-1}W_{22}^{t-1}$$

and where W_{ii}^t is W_{ii} raised to the tth power.

Write equation (5.15) as

$$\begin{pmatrix} y_{1t}^* \\ y_{2t}^* \end{pmatrix} = \begin{bmatrix} W_{11}^t & W_{12,t} \\ 0 & W_{22}^t \end{bmatrix} \begin{pmatrix} y_{10}^* \\ y_{20}^* \end{pmatrix}$$

where $y_t^* = V^{-1}y_t$, and in particular where

$$y_{2t}^* = V^{21}x_t + V^{22}\mu_t, \tag{5.16}$$

and where V^{ij} denotes the (i,j) piece of the partitioned V^{-1} matrix.

Because W_{22} is an unstable matrix, y_t^* will diverge unless $y_{20}^* = 0$.

Let V^{ij} denote the (i,j) piece of the partitioned V^{-1} matrix.

To attain stability, we must impose $y_{20}^* = 0$, which from equation (5.16) implies

$$V^{21}x_0 + V^{22}\mu_0 = 0$$

or

$$\mu_0 = -(V^{22})^{-1}V^{21}x_0.$$

This equation replicates itself over time in the sense that it implies

$$\mu_t = -(V^{22})^{-1} V^{21} x_t.$$

But notice that because $(V^{21} \; V^{22})$ is the second row block of the inverse of V, it follows that

$$(V^{21} \; V^{22}) \quad \begin{pmatrix} V_{11} \\ V_{21} \end{pmatrix} = 0$$

which implies

$$V^{21} V_{11} + V^{22} V_{21} = 0.$$

Therefore,

$$-(V^{22})^{-1} V^{21} = V_{21} V_{11}^{-1}.$$

So we can write

$$\mu_0 = V_{21} V_{11}^{-1} x_0$$

and

$$\mu_t = V_{21} V_{11}^{-1} x_t.$$

However, we know that $\mu_t = P x_t$, where P occurs in the matrix that solves the Riccati equation.

Thus, the preceding argument establishes that

$$P = V_{21} V_{11}^{-1}. \tag{5.17}$$

Remarkably, formula (5.17) provides us with a computationally efficient way of computing the positive definite matrix P that solves the algebraic Riccati equation (5.2) that emerges from dynamic programming.

This same method can be applied to compute the solution of any system of the form (5.8) if a solution exists, even if eigenvalues of M fail to occur in reciprocal pairs.

The method will typically work so long as the eigenvalues of M split half inside and half outside the unit circle.

Systems in which eigenvalues (properly adjusted for discounting) fail to occur in reciprocal pairs arise when the system being solved is an equilibrium of a model in which there are distortions that prevent there being any optimum problem that the equilibrium solves. See [Ljungqvist and Sargent, 2018], ch 12.

5.7 Application

Here we demonstrate the computation with an example which is the deterministic version of an example borrowed from this quantecon lecture.

```
# Model parameters
r = 0.05
c_bar = 2
μ = 1

# Formulate as an LQ problem
Q = np.array([[1]])
```

(continues on next page)

(continued from previous page)

```
R = np.zeros((2, 2))
A = [[1 + r, -c_bar + μ],
     [0,              1]]
B = [[-1],
     [0]]

# Construct an LQ instance
lq = LQ(Q, R, A, B)
```

Given matrices A, B, Q, R, we can then compute L, N, and $M = L^{-1}N$.

```
def construct_LNM(A, B, Q, R):

    n, k = lq.n, lq.k

    # construct L and N
    L = np.zeros((2*n, 2*n))
    L[:n, :n] = np.eye(n)
    L[:n, n:] = B @ np.linalg.inv(Q) @ B.T
    L[n:, n:] = A.T

    N = np.zeros((2*n, 2*n))
    N[:n, :n] = A
    N[n:, :n] = -R
    N[n:, n:] = np.eye(n)

    # compute M
    M = np.linalg.inv(L) @ N

    return L, N, M
```

```
L, N, M = construct_LNM(lq.A, lq.B, lq.Q, lq.R)
```

```
M
```

```
array([[ 1.05      , -1.        , -0.95238095,  0.        ],
       [ 0.        ,  1.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.95238095,  0.        ],
       [ 0.        ,  0.        ,  0.95238095,  1.        ]])
```

Let's verify that M is symplectic.

```
n = lq.n
J = np.zeros((2*n, 2*n))
J[n:, :n] = np.eye(n)
J[:n, n:] = -np.eye(n)

M @ J @ M.T - J
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

We can compute the eigenvalues of M using `np.linalg.eigvals`, arranged in ascending order.

```
eigvals = sorted(np.linalg.eigvals(M))
eigvals
```

```
[0.9523809523809523, 1.0, 1.0, 1.05]
```

When we apply Schur decomposition such that $M = VWV^{-1}$, we want

- the upper left block of W, W_{11}, to have all of its eigenvalues less than 1 in modulus, and
- the lower right block W_{22} to have eigenvalues that exceed 1 in modulus.

To get what we want, let's define a sorting function that tells `scipy.schur` to sort the corresponding eigenvalues with modulus smaller than 1 to the upper left.

```
stable_eigvals = eigvals[:n]

def sort_fun(x):
    "Sort the eigenvalues with modules smaller than 1 to the top-left."

    if x in stable_eigvals:
        stable_eigvals.pop(stable_eigvals.index(x))
        return True
    else:
        return False

W, V, _ = schur(M, sort=sort_fun)
```

```
W
```

```
array([[ 1.        , -0.02316402, -1.00085948, -0.95000594],
       [ 0.        ,  0.95238095, -0.00237501, -0.95325452],
       [ 0.        ,  0.        ,  1.05      ,  0.02432222],
       [ 0.        ,  0.        ,  0.        ,  1.        ]])
```

```
V
```

```
array([[ 0.99875234,  0.00121459, -0.04992284,  0.        ],
       [ 0.04993762, -0.02429188,  0.99845688,  0.        ],
       [ 0.        ,  0.04992284,  0.00121459,  0.99875234],
       [ 0.        , -0.99845688, -0.02429188,  0.04993762]])
```

We can check the modulus of eigenvalues of W_{11} and W_{22}.

Since they are both triangular matrices, eigenvalues are the diagonal elements.

```
# W11
np.diag(W[:n, :n])
```

```
array([1.        , 0.95238095])
```

```
# W22
np.diag(W[n:, n:])
```

```
array([1.05, 1.  ])
```

The following functions wrap M matrix construction, Schur decomposition, and stability-imposing computation of P.

```python
def stable_solution(M, verbose=True):
    """
    Given a system of linear difference equations

        y' = |a b| y
        x' = |c d| x

    which is potentially unstable, find the solution
    by imposing stability.

    Parameter
    ---------
    M : np.ndarray(float)
        The matrix represents the linear difference equations system.
    """
    n = M.shape[0] // 2
    stable_eigvals = list(sorted(np.linalg.eigvals(M))[:n])

    def sort_fun(x):
        "Sort the eigenvalues with modules smaller than 1 to the top-left."

        if x in stable_eigvals:
            stable_eigvals.pop(stable_eigvals.index(x))
            return True
        else:
            return False

    W, V, _ = schur(M, sort=sort_fun)
    if verbose:
        print('eigenvalues:\n')
        print('    W11: {}'.format(np.diag(W[:n, :n])))
        print('    W22: {}'.format(np.diag(W[n:, n:])))

    # compute V21 V11^{-1}
    P = V[n:, :n] @ np.linalg.inv(V[:n, :n])

    return W, V, P

def stationary_P(lq, verbose=True):
    """
    Computes the matrix :math:`P` that represent the value function

         V(x) = x' P x

    in the infinite horizon case. Computation is via imposing stability
    on the solution path and using Schur decomposition.

    Parameters
    ----------
    lq : qe.LQ
        QuantEcon class for analyzing linear quadratic optimal control
        problems of infinite horizon form.
```

(continues on next page)

(continued from previous page)

```
    Returns
    -------
    P : array_like(float)
        P matrix in the value function representation.
    """

    Q = lq.Q
    R = lq.R
    A = lq.A * lq.beta ** (1/2)
    B = lq.B * lq.beta ** (1/2)

    n, k = lq.n, lq.k

    L, N, M = construct_LNM(A, B, Q, R)
    W, V, P = stable_solution(M, verbose=verbose)

    return P
```

```
# compute P
stationary_P(lq)
```

```
eigenvalues:

    W11: [1.         0.95238095]
    W22: [1.05 1.  ]
```

```
array([[ 0.1025, -2.05  ],
       [-2.05  , 41.    ]])
```

Note that the matrix P computed in this way is close to what we get from the routine in quantecon that solves an algebraic Riccati equation by iterating to convergence on a Riccati difference equation.

The small difference comes from computational errors and will decrease as we increase the maximum number of iterations or decrease the tolerance for convergence.

```
lq.stationary_values()
```

```
(array([[ 0.1025, -2.05  ],
        [-2.05  , 41.01  ]]),
 array([[-0.09761905,  1.95238095]]),
 0)
```

Using a Schur decomposition is much more efficient.

```
%%timeit
stationary_P(lq, verbose=False)
```

```
72.4 µs ± 241 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

```
%%timeit
lq.stationary_values()
```

```
1.16 ms ± 2.52 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

5.8 Other Applications

The preceding approach to imposing stability on a system of potentially unstable linear difference equations is not limited to linear quadratic dynamic optimization problems.

For example, the same method is used in our Stability in Linear Rational Expectations Models lecture.

Let's try to solve the model described in that lecture by applying the `stable_solution` function defined in this lecture above.

```
def construct_H(ρ, λ, δ):
    "contruct matrix H given parameters."

    H = np.empty((2, 2))
    H[0, :] = ρ,δ
    H[1, :] = - (1 - λ) / λ, 1 / λ

    return H

H = construct_H(ρ=.9, λ=.5, δ=0)
```

```
W, V, P = stable_solution(H)
P
```

```
eigenvalues:

    W11: [0.9]
    W22: [2.]
```

```
array([[0.90909091]])
```

5.9 Discounted Problems

5.9.1 Transforming States and Controls to Eliminate Discounting

A pair of useful transformations allows us to convert a discounted problem into an undiscounted one.

Thus, suppose that we have a discounted problem with objective

$$-\sum_{t=0}^{\infty} \beta^t \left\{ x_t' R x_t + u_t' Q u_t \right\}$$

and that the state transition equation is again $x_{t+1} = Ax_t + Bu_t$.

Define the transformed state and control variables

- $\hat{x}_t = \beta^{\frac{t}{2}} x_t$
- $\hat{u}_t = \beta^{\frac{t}{2}} u_t$

and the transformed transition equation matrices

- $\hat{A} = \beta^{\frac{1}{2}} A$
- $\hat{B} = \beta^{\frac{1}{2}} B$

so that the adjusted state and control variables obey the transition law

$$\hat{x}_{t+1} = \hat{A}\hat{x}_t + \hat{B}\hat{u}_t.$$

Then a discounted optimal control problem defined by A, B, R, Q, β having optimal policy characterized by P, F is associated with an equivalent undiscounted problem defined by $\hat{A}, \hat{B}, Q, R$ having optimal policy characterized by $\hat{F}, \hat{P}$ that satisfy the following equations:

$$\hat{F} = (Q + B'\hat{P}B)^{-1}\hat{B}'P\hat{A}$$

and

$$\hat{P} = R + \hat{A}'P\hat{A} - \hat{A}'P\hat{B}(Q + B'\hat{P}\hat{B})^{-1}\hat{B}'P\hat{A}$$

It follows immediately from the definitions of $\hat{A}, \hat{B}$ that $\hat{F} = F$ and $\hat{P} = P$.

By exploiting these transformations, we can solve a discounted problem by solving an associated undiscounted problem.

In particular, we can first transform a discounted LQ problem to an undiscounted one and then solve that discounted optimal regulator problem using the Lagrangian and invariant subspace methods described above.

For example, when $\beta = \frac{1}{1+r}$, we can solve for P with $\hat{A} = \beta^{1/2} A$ and $\hat{B} = \beta^{1/2} B$.

These settings are adopted by default in the function `stationary_P` defined above.

```
β = 1 / (1 + r)
lq.beta = β
```

```
stationary_P(lq)
```

```
eigenvalues:

    W11: [0.97590007 0.97590007]
    W22: [1.02469508 1.02469508]
```

```
array([[ 0.0525, -1.05  ],
       [-1.05  , 21.    ]])
```

We can verify that the solution agrees with one that comes from applying the routine `LQ.stationary_values` in the quantecon package.

```
lq.stationary_values()
```

```
(array([[ 0.0525, -1.05  ],
        [-1.05  , 21.    ]]),
 array([[-0.05,  1.  ]]),
 0.0)
```

5.9.2 Lagrangian for Discounted Problem

For several purposes, it is useful explicitly briefly to describe a Lagrangian for a discounted problem.

Thus, for the discounted optimal linear regulator problem, form the Lagrangian

$$L = -\sum_{t=0}^{\infty} \beta^t \left\{ x_t' R x_t + u_t' Q u_t + 2\beta \mu_{t+1}' [A x_t + B u_t - x_{t+1}] \right\} \tag{5.18}$$

where $2\mu_{t+1}$ is a vector of Lagrange multipliers on the state vector x_{t+1}.

First-order conditions for maximization with respect to $\{u_t, x_{t+1}\}_{t=0}^{\infty}$ are

$$\begin{aligned} 2Qu_t + 2\beta B' \mu_{t+1} &= 0 \ , \ t \geq 0 \\ \mu_t &= R x_t + \beta A' \mu_{t+1} \ , \ t \geq 1. \end{aligned} \tag{5.19}$$

Define $2\mu_0$ to be the vector of shadow prices of x_0 and apply an envelope condition to (5.18) to deduce that

$$\mu_0 = R x_0 + \beta A' \mu_1,$$

which is a time $t = 0$ counterpart to the second equation of system (5.19).

Proceeding as we did above with the undiscounted system (5.5), we can rearrange the first-order conditions into the system

$$\begin{bmatrix} I & \beta B Q^{-1} B' \\ 0 & \beta A' \end{bmatrix} \begin{bmatrix} x_{t+1} \\ \mu_{t+1} \end{bmatrix} = \begin{bmatrix} A & 0 \\ -R & I \end{bmatrix} \begin{bmatrix} x_t \\ \mu_t \end{bmatrix} \tag{5.20}$$

which in the special case that $\beta = 1$ agrees with equation (5.5), as expected.

By staring at system (5.20), we can infer identities that shed light on the structure of optimal linear regulator problems, some of which will be useful in this lecture when we apply and extend the methods of this lecture to study Stackelberg and Ramsey problems.

First, note that the first block of equation system (5.20) asserts that when $\mu_{t+1} = P x_{t+1}$, then

$$(I + \beta Q^{-1} B' P B P) x_{t+1} = A x_t,$$

which can be rearranged to sbe

$$x_{t+1} = (I + \beta B Q^{-1} B' P)^{-1} A x_t.$$

This expression for the optimal closed loop dynamics of the state must agree with an alternative expression that we had derived with dynamic programming, namely,

$$x_{t+1} = (A - BF) x_t.$$

But using

$$F = \beta (Q + \beta B' P B)^{-1} B' P A \tag{5.21}$$

it follows that

$$A - BF = (I - \beta B (Q + \beta B' P B)^{-1} B' P) A.$$

Thus, our two expressions for the closed loop dynamics agree if and only if

$$(I + \beta B Q^{-1} B' P)^{-1} = (I - \beta B (Q + \beta B' P B)^{-1} B' P). \tag{5.22}$$

Matrix equation (5.22) can be verified by applying a partitioned inverse formula.

Note: Just use the formula $(a - bd^{-1}c)^{-1} = a^{-1} + a^{-1}b(d - ca^{-1}b)^{-1}ca^{-1}$ for appropriate choices of the matrices a, b, c, d.

Next, note that for *any* fixed F for which eigenvalues of $A - BF$ are less than $\frac{1}{\beta}$ in modulus, the value function associated with using this rule forever is $-x_0 \tilde{P} x_0$ where $\tilde{P}$ obeys the following matrix equation:

$$\tilde{P} = (R + F'QF) + \beta(A - BF)'P(A - BF). \tag{5.23}$$

Evidently, $\tilde{P} = P$ only when F obeys formula (5.21).

Next, note that the second equation of system (5.20) implies the "forward looking" equation for the Lagrange multiplier

$$\mu_t = Rx_t + \beta A' \mu_{t+1}$$

whose solution is

$$\mu_t = Px_t,$$

where

$$P = R + \beta A'P(A - BF) \tag{5.24}$$

where we must require that F obeys equation (5.21).

Equations (5.23) and (5.24) provide different perspectives on the optimal value function.

CHAPTER

SIX

ELIMINATING CROSS PRODUCTS

6.1 Overview

This lecture describes formulas for eliminating

- cross products between states and control in linear-quadratic dynamic programming problems
- covariances between state and measurement noises in Kalman filtering problems

For a linear-quadratic dynamic programming problem, the idea involves these steps

- transform states and controls in a way that leads to an equivalent problem with no cross-products between transformed states and controls
- solve the transformed problem using standard formulas for problems with no cross-products between states and controls presented in this lecture *Linear Control: Foundations*
- transform the optimal decision rule for the altered problem into the optimal decision rule for the original problem with cross-products between states and controls

6.2 Undiscounted Dynamic Programming Problem

Here is a nonstochastic undiscounted LQ dynamic programming with cross products between states and controls in the objective function.

The problem is defined by the 5-tuple of matrices (A, B, R, Q, H) where R and Q are positive definite symmetric matrices and $A \sim m \times m, B \sim m \times k, Q \sim k \times k, R \sim m \times m$ and $H \sim k \times m$.

The problem is to choose $\{x_{t+1}, u_t\}_{t=0}^{\infty}$ to maximize

$$
-\sum_{t=0}^{\infty}(x_t' R x_t + u_t' Q u_t + 2u_t H x_t)
$$

subject to the linear constraints

$$
x_{t+1} = Ax_t + Bu_t, \quad t \geq 0
$$

where x_0 is a given initial condition.

The solution to this undiscounted infinite-horizon problem is a time-invariant feedback rule

$$
u_t = -Fx_t
$$

where

$$F = -(Q + B'PB)^{-1}B'PA$$

and $P \sim m \times m$ is a positive definite solution of the algebraic matrix Riccati equation

$$P = R + A'PA - (A'PB + H')(Q + B'PB)^{-1}(B'PA + H).$$

It can be verified that an **equivalent** problem without cross-products between states and controls is defined by a 4-tuple of matrices : (A^*, B, R^*, Q).

That the omitted matrix $H = 0$ indicates that there are no cross products between states and controls in the equivalent problem.

The matrices (A^*, B, R^*, Q) defining the equivalent problem and the value function, policy function matrices P, F^* that solve it are related to the matrices (A, B, R, Q, H) defining the original problem and the value function, policy function matrices P, F that solve the original problem by

$$\begin{aligned}
A^* &= A - BQ^{-1}H, \\
R^* &= R - H'Q^{-1}H, \\
P &= R^* + A^{*\prime}PA - (A^{*\prime}PB)(Q + B'PB)^{-1}B'PA^*, \\
F^* &= (Q + B'PB)^{-1}B'PA^*, \\
F &= F^* + Q^{-1}H.
\end{aligned}$$

6.3 Kalman Filter

The **duality** that prevails between a linear-quadratic optimal control and a Kalman filtering problem means that there is an analogous transformation that allows us to transform a Kalman filtering problem with non-zero covariance matrix between between shocks to states and shocks to measurements to an equivalent Kalman filtering problem with zero covariance between shocks to states and measurments.

Let's look at the appropriate transformations.

First, let's recall the Kalman filter with covariance between noises to states and measurements.

The hidden Markov model is

$$\begin{aligned}
x_{t+1} &= Ax_t + Bw_{t+1}, \\
z_{t+1} &= Dx_t + Fw_{t+1},
\end{aligned}$$

where $A \sim m \times m, B \sim m \times p$ and $D \sim k \times m, F \sim k \times p$, and w_{t+1} is the time $t + 1$ component of a sequence of i.i.d. $p \times 1$ normally distibuted random vectors with mean vector zero and covariance matrix equal to a $p \times p$ identity matrix.

Thus, x_t is $m \times 1$ and z_t is $k \times 1$.

The Kalman filtering formulas are

$$\begin{aligned}
K(\Sigma_t) &= (A\Sigma_t D' + BF')(D\Sigma_t D' + FF')^{-1}, \\
\Sigma_{t+1} &= A\Sigma_t A' + BB' - (A\Sigma_t D' + BF')(D\Sigma_t D' + FF')^{-1}(D\Sigma_t A' + FB').
\end{aligned}$$

Define tranformed matrices

$$\begin{aligned}
A^* &= A - BF'(FF')^{-1}D, \\
B^*B^{*\prime} &= BB' - BF'(FF')^{-1}FB'.
\end{aligned}$$

6.3.1 Algorithm

A consequence of formulas {eq}`eq:Kalman102} is that we can use the following algorithm to solve Kalman filtering problems that involve non zero covariances between state and signal noises.

First, compute Σ, K^* using the ordinary Kalman filtering formula with $BF' = 0$, i.e., with zero covariance matrix between random shocks to states and random shocks to measurements.

That is, compute K^* and Σ that satisfy

$$\begin{aligned} K^* &= (A^*\Sigma D')(D\Sigma D' + FF')^{-1} \\ \Sigma &= A^*\Sigma A^{*'} + B^*B^{*'} - (A^*\Sigma D')(D\Sigma D' + FF')^{-1}(D\Sigma A^{*'}). \end{aligned}$$

The Kalman gain for the original problem **with non-zero covariance** between shocks to states and measurements is then

$$K = K^* + BF'(FF')^{-1},$$

The state reconstruction covariance matrix Σ for the original problem equals the state reconstrution covariance matrix for the transformed problem.

6.4 Duality table

Here is a handy table to remember how the Kalman filter and dynamic program are related.

Dynamic Program	Kalman Filter
A	A'
B	D'
H	FB'
Q	FF'
R	BB'
F	K'
P	Σ

CHAPTER

SEVEN

THE PERMANENT INCOME MODEL

Contents

7.1 Overview

This lecture describes a rational expectations version of the famous permanent income model of Milton Friedman [Friedman, 1956].

Robert Hall cast Friedman's model within a linear-quadratic setting [Hall, 1978].

Like Hall, we formulate an infinite-horizon linear-quadratic savings problem.

We use the model as a vehicle for illustrating

- alternative formulations of the *state* of a dynamic system
- the idea of *cointegration*
- impulse response functions
- the idea that changes in consumption are useful as predictors of movements in income

Background readings on the linear-quadratic-Gaussian permanent income model are Hall's [Hall, 1978] and chapter 2 of [Ljungqvist and Sargent, 2018].

Let's start with some imports

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
import random
from numba import njit
```

7.2 The Savings Problem

In this section, we state and solve the savings and consumption problem faced by the consumer.

7.2.1 Preliminaries

We use a class of stochastic processes called martingales.

A discrete-time martingale is a stochastic process (i.e., a sequence of random variables) $\{X_t\}$ with finite mean at each t and satisfying

$$\mathbb{E}_t[X_{t+1}] = X_t, \qquad t = 0, 1, 2, \ldots$$

Here $\mathbb{E}_t := \mathbb{E}[\cdot \mid \mathcal{F}_t]$ is a conditional mathematical expectation conditional on the time t *information set* $\mathcal{F}_t$.

The latter is just a collection of random variables that the modeler declares to be visible at t.

- When not explicitly defined, it is usually understood that $\mathcal{F}_t = \{X_t, X_{t-1}, \ldots, X_0\}$.

Martingales have the feature that the history of past outcomes provides no predictive power for changes between current and future outcomes.

For example, the current wealth of a gambler engaged in a "fair game" has this property.

One common class of martingales is the family of *random walks*.

A **random walk** is a stochastic process $\{X_t\}$ that satisfies

$$X_{t+1} = X_t + w_{t+1}$$

for some IID zero mean *innovation* sequence $\{w_t\}$.

Evidently, X_t can also be expressed as

$$X_t = \sum_{j=1}^{t} w_j + X_0$$

Not every martingale arises as a random walk (see, for example, Wald's martingale).

7.2.2 The Decision Problem

A consumer has preferences over consumption streams that are ordered by the utility functional

$$\mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t) \right] \tag{7.1}$$

where

- $\mathbb{E}_t$ is the mathematical expectation conditioned on the consumer's time t information
- c_t is time t consumption
- u is a strictly concave one-period utility function
- $\beta \in (0, 1)$ is a discount factor

The consumer maximizes (7.1) by choosing a consumption, borrowing plan $\{c_t, b_{t+1}\}_{t=0}^\infty$ subject to the sequence of budget constraints

$$c_t + b_t = \frac{1}{1+r} b_{t+1} + y_t \quad t \geq 0 \tag{7.2}$$

Here

- y_t is an exogenous endowment process.
- $r > 0$ is a time-invariant risk-free net interest rate.
- b_t is one-period risk-free debt maturing at t.

The consumer also faces initial conditions b_0 and y_0, which can be fixed or random.

7.2.3 Assumptions

For the remainder of this lecture, we follow Friedman and Hall in assuming that $(1+r)^{-1} = \beta$.

Regarding the endowment process, we assume it has the *state-space representation*

$$\begin{aligned} z_{t+1} &= A z_t + C w_{t+1} \\ y_t &= U z_t \end{aligned} \tag{7.3}$$

where

- $\{w_t\}$ is an IID vector process with $\mathbb{E} w_t = 0$ and $\mathbb{E} w_t w_t' = I$.
- The spectral radius of A satisfies $\rho(A) < \sqrt{1/\beta}$.
- U is a selection vector that pins down y_t as a particular linear combination of components of z_t.

The restriction on $\rho(A)$ prevents income from growing so fast that discounted geometric sums of some quadratic forms to be described below become infinite.

Regarding preferences, we assume the quadratic utility function

$$u(c_t) = -(c_t - \gamma)^2$$

where γ is a bliss level of consumption.

Note: Along with this quadratic utility specification, we allow consumption to be negative. However, by choosing parameters appropriately, we can make the probability that the model generates negative consumption paths over finite time horizons as low as desired.

Finally, we impose the *no Ponzi scheme* condition

$$\mathbb{E}_0 \left[\sum_{t=0}^\infty \beta^t b_t^2 \right] < \infty \tag{7.4}$$

This condition rules out an always-borrow scheme that would allow the consumer to enjoy bliss consumption forever.

7.2.4 First-Order Conditions

First-order conditions for maximizing (7.1) subject to (7.2) are

$$\mathbb{E}_t[u'(c_{t+1})] = u'(c_t), \qquad t = 0, 1, ... \tag{7.5}$$

These optimality conditions are also known as *Euler equations*.

If you're not sure where they come from, you can find a proof sketch in the *appendix*.

With our quadratic preference specification, (7.5) has the striking implication that consumption follows a martingale:

$$\mathbb{E}_t[c_{t+1}] = c_t \tag{7.6}$$

(In fact, quadratic preferences are *necessary* for this conclusion[1].)

One way to interpret (7.6) is that consumption will change only when "new information" about permanent income is revealed.

These ideas will be clarified below.

7.2.5 The Optimal Decision Rule

Now let's deduce the optimal decision rule[2].

Note: One way to solve the consumer's problem is to apply *dynamic programming* as in *this lecture*. We do this later. But first we use an alternative approach that is revealing and shows the work that dynamic programming does for us behind the scenes.

In doing so, we need to combine

1. the optimality condition (7.6)
2. the period-by-period budget constraint (7.2), and
3. the boundary condition (7.4)

To accomplish this, observe first that (7.4) implies $\lim_{t\to\infty} \beta^{\frac{t}{2}} b_{t+1} = 0$.

Using this restriction on the debt path and solving (7.2) forward yields

$$b_t = \sum_{j=0}^{\infty} \beta^j (y_{t+j} - c_{t+j}) \tag{7.7}$$

Take conditional expectations on both sides of (7.7) and use the martingale property of consumption and the *law of iterated expectations* to deduce

$$b_t = \sum_{j=0}^{\infty} \beta^j \mathbb{E}_t[y_{t+j}] - \frac{c_t}{1-\beta} \tag{7.8}$$

Expressed in terms of c_t we get

$$c_t = (1-\beta)\left[\sum_{j=0}^{\infty} \beta^j \mathbb{E}_t[y_{t+j}] - b_t\right] = \frac{r}{1+r}\left[\sum_{j=0}^{\infty} \beta^j \mathbb{E}_t[y_{t+j}] - b_t\right] \tag{7.9}$$

[1] A linear marginal utility is essential for deriving (7.6) from (7.5). Suppose instead that we had imposed the following more standard assumptions on the utility function: $u'(c) > 0, u''(c) < 0, u'''(c) > 0$ and required that $c \geq 0$. The Euler equation remains (7.5). But the fact that $u''' < 0$ implies via Jensen's inequality that $\mathbb{E}_t[u'(c_{t+1})] > u'(\mathbb{E}_t[c_{t+1}])$. This inequality together with (7.5) implies that $\mathbb{E}_t[c_{t+1}] > c_t$ (consumption is said to be a 'submartingale'), so that consumption stochastically diverges to $+\infty$. The consumer's savings also diverge to $+\infty$.

[2] An optimal decision rule is a map from the current state into current actions—in this case, consumption.

where the last equality uses $(1+r)\beta = 1$.

These last two equations assert that consumption equals *economic income*

- **financial wealth** equals $-b_t$
- **non-financial wealth** equals $\sum_{j=0}^{\infty} \beta^j \mathbb{E}_t[y_{t+j}]$
- **total wealth** equals the sum of financial and non-financial wealth
- a **marginal propensity to consume out of total wealth** equals the interest factor $\frac{r}{1+r}$
- **economic income** equals
  - a constant marginal propensity to consume times the sum of non-financial wealth and financial wealth
  - the amount the consumer can consume while leaving its wealth intact

Responding to the State

The *state* vector confronting the consumer at t is $\begin{bmatrix} b_t & z_t \end{bmatrix}$.

Here

- z_t is an *exogenous* component, unaffected by consumer behavior.
- b_t is an *endogenous* component (since it depends on the decision rule).

Note that z_t contains all variables useful for forecasting the consumer's future endowment.

It is plausible that current decisions c_t and b_{t+1} should be expressible as functions of z_t and b_t.

This is indeed the case.

In fact, from *this discussion*, we see that

$$\sum_{j=0}^{\infty} \beta^j \mathbb{E}_t[y_{t+j}] = \mathbb{E}_t \left[\sum_{j=0}^{\infty} \beta^j y_{t+j} \right] = U(I - \beta A)^{-1} z_t$$

Combining this with (7.9) gives

$$c_t = \frac{r}{1+r} \left[U(I - \beta A)^{-1} z_t - b_t \right] \tag{7.10}$$

Using this equality to eliminate c_t in the budget constraint (7.2) gives

$$\begin{aligned} b_{t+1} &= (1+r)(b_t + c_t - y_t) \\ &= (1+r)b_t + r[U(I-\beta A)^{-1} z_t - b_t] - (1+r)U z_t \\ &= b_t + U[r(I-\beta A)^{-1} - (1+r)I] z_t \\ &= b_t + U(I-\beta A)^{-1}(A - I) z_t \end{aligned}$$

To get from the second last to the last expression in this chain of equalities is not trivial.

A key is to use the fact that $(1+r)\beta = 1$ and $(I - \beta A)^{-1} = \sum_{j=0}^{\infty} \beta^j A^j$.

We've now successfully written c_t and b_{t+1} as functions of b_t and z_t.

A State-Space Representation

We can summarize our dynamics in the form of a linear state-space system governing consumption, debt and income:

$$\begin{aligned} z_{t+1} &= A z_t + C w_{t+1} \\ b_{t+1} &= b_t + U[(I - \beta A)^{-1} (A - I)] z_t \\ y_t &= U z_t \\ c_t &= (1-\beta)[U(I - \beta A)^{-1} z_t - b_t] \end{aligned} \tag{7.11}$$

To write this more succinctly, let

$$x_t = \begin{bmatrix} z_t \\ b_t \end{bmatrix}, \quad \tilde A = \begin{bmatrix} A & 0 \\ U(I-\beta A)^{-1}(A-I) & 1 \end{bmatrix}, \quad \tilde C = \begin{bmatrix} C \\ 0 \end{bmatrix}$$

and

$$\tilde U = \begin{bmatrix} U & 0 \\ (1-\beta)U(I-\beta A)^{-1} & -(1-\beta) \end{bmatrix}, \quad \tilde y_t = \begin{bmatrix} y_t \\ c_t \end{bmatrix}$$

Then we can express equation (7.11) as

$$\begin{aligned} x_{t+1} &= \tilde A x_t + \tilde C w_{t+1} \\ \tilde y_t &= \tilde U x_t \end{aligned} \tag{7.12}$$

We can use the following formulas from *linear state space models* to compute population mean $\mu_t = \mathbb{E} x_t$ and covariance $\Sigma_t := \mathbb{E}[(x_t - \mu_t)(x_t - \mu_t)']$

$$\mu_{t+1} = \tilde A \mu_t \quad \text{with} \quad \mu_0 \text{ given} \tag{7.13}$$

$$\Sigma_{t+1} = \tilde A \Sigma_t \tilde A' + \tilde C \tilde C' \quad \text{with} \quad \Sigma_0 \text{ given} \tag{7.14}$$

We can then compute the mean and covariance of $\tilde y_t$ from

$$\begin{aligned} \mu_{y,t} &= \tilde U \mu_t \\ \Sigma_{y,t} &= \tilde U \Sigma_t \tilde U' \end{aligned} \tag{7.15}$$

A Simple Example with IID Income

To gain some preliminary intuition on the implications of (7.11), let's look at a highly stylized example where income is just IID.

(Later examples will investigate more realistic income streams.)

In particular, let $\{w_t\}_{t=1}^{\infty}$ be IID and scalar standard normal, and let

$$z_t = \begin{bmatrix} z_t^1 \\ 1 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & \mu \end{bmatrix}, \quad C = \begin{bmatrix} \sigma \\ 0 \end{bmatrix}$$

Finally, let $b_0 = z_0^1 = 0$.

Under these assumptions, we have $y_t = \mu + \sigma w_t \sim N(\mu, \sigma^2)$.

Further, if you work through the state space representation, you will see that

$$b_t = -\sigma \sum_{j=1}^{t-1} w_j$$

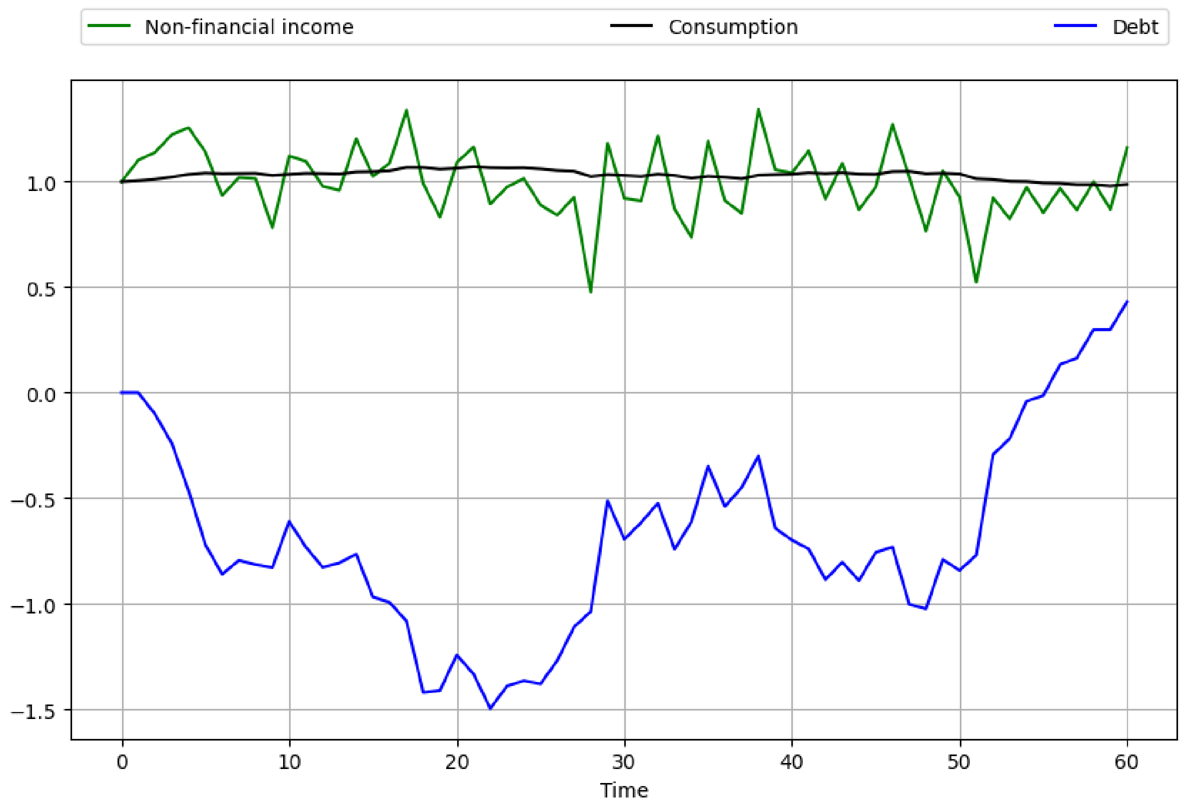
$$c_t = \mu + (1-\beta)\sigma \sum_{j=1}^{t} w_j$$

Thus, income is IID and debt and consumption are both Gaussian random walks.

Defining assets as $-b_t$, we see that assets are just the cumulative sum of unanticipated incomes prior to the present date.

The next figure shows a typical realization with $r = 0.05$, $\mu = 1$, and $\sigma = 0.15$

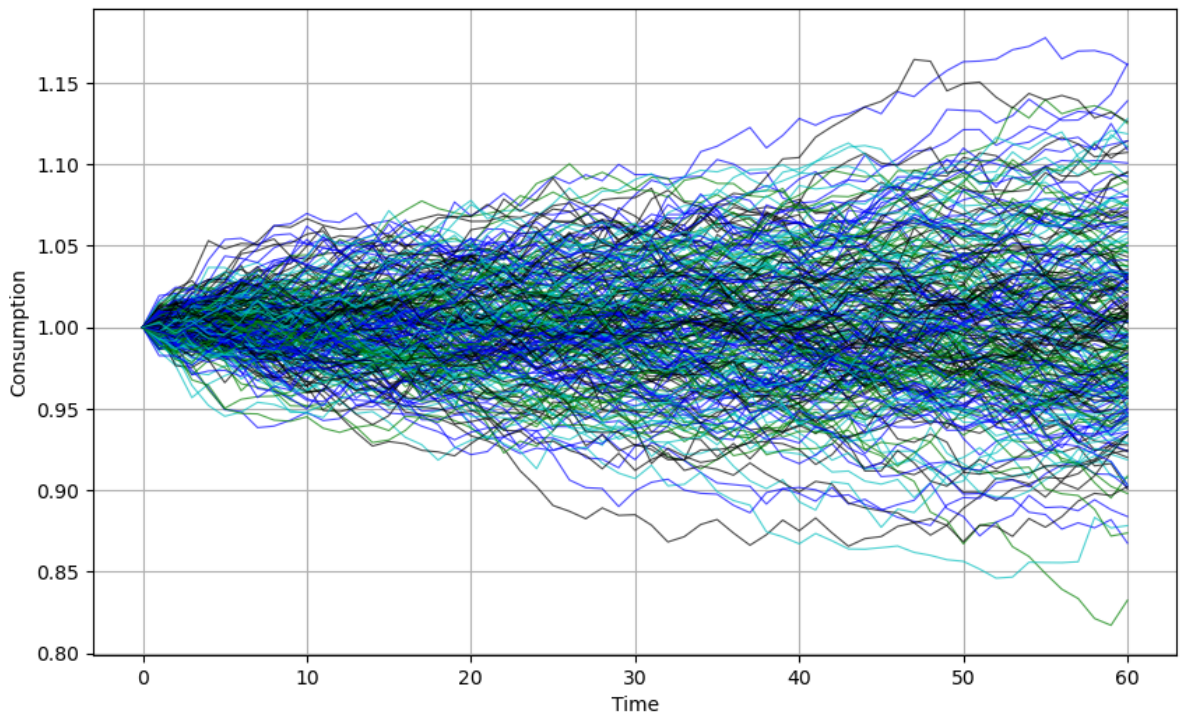
```
r = 0.05
β = 1 / (1 + r)
σ = 0.15
μ = 1
T = 60

@njit
def time_path(T):
    w = np.random.randn(T+1)  # w_0, w_1, ..., w_T
    w[0] = 0
    b = np.zeros(T+1)
    for t in range(1, T+1):
        b[t] = w[1:t].sum()
    b = -σ * b
    c = μ + (1 - β) * (σ * w - b)
    return w, b, c

w, b, c = time_path(T)

fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(μ + σ * w, 'g-', label="Non-financial income")
ax.plot(c, 'k-', label="Consumption")
ax.plot( b, 'b-', label="Debt")
ax.legend(ncol=3, mode='expand', bbox_to_anchor=(0., 1.02, 1., .102))
ax.grid()
ax.set_xlabel('Time')

plt.show()
```



Observe that consumption is considerably smoother than income.

The figure below shows the consumption paths of 250 consumers with independent income streams

```
fig, ax = plt.subplots(figsize=(10, 6))

b_sum = np.zeros(T+1)
for i in range(250):
    w, b, c = time_path(T)  # Generate new time path
    rcolor = random.choice(('c', 'g', 'b', 'k'))
    ax.plot(c, color=rcolor, lw=0.8, alpha=0.7)

ax.grid()
ax.set(xlabel='Time', ylabel='Consumption')

plt.show()
```



7.3 Alternative Representations

In this section, we shed more light on the evolution of savings, debt and consumption by representing their dynamics in several different ways.

7.3.1 Hall's Representation

Hall [Hall, 1978] suggested an insightful way to summarize the implications of LQ permanent income theory.

First, to represent the solution for b_t, shift (7.9) forward one period and eliminate b_{t+1} by using (7.2) to obtain

$$c_{t+1} = (1-\beta)\sum_{j=0}^{\infty}\beta^j \mathbb{E}_{t+1}[y_{t+j+1}] - (1-\beta)\left[\beta^{-1}(c_t + b_t - y_t)\right]$$

If we add and subtract $\beta^{-1}(1-\beta)\sum_{j=0}^{\infty}\beta^j \mathbb{E}_t y_{t+j}$ from the right side of the preceding equation and rearrange, we obtain

$$c_{t+1} - c_t = (1-\beta)\sum_{j=0}^{\infty}\beta^j \left\{\mathbb{E}_{t+1}[y_{t+j+1}] - \mathbb{E}_t[y_{t+j+1}]\right\} \tag{7.16}$$

The right side is the time $t+1$ *innovation to the expected present value* of the endowment process $\{y_t\}$.

We can represent the optimal decision rule for (c_t, b_{t+1}) in the form of (7.16) and (7.8), which we repeat:

$$b_t = \sum_{j=0}^{\infty}\beta^j \mathbb{E}_t[y_{t+j}] - \frac{1}{1-\beta}c_t \tag{7.17}$$

Equation (7.17) asserts that the consumer's debt due at t equals the expected present value of its endowment minus the expected present value of its consumption stream.

A high debt thus indicates a large expected present value of surpluses $y_t - c_t$.

Recalling again our discussion on *forecasting geometric sums*, we have

$$\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} = U(I - \beta A)^{-1} z_t$$

$$\mathbb{E}_{t+1} \sum_{j=0}^{\infty} \beta^j y_{t+j+1} = U(I - \beta A)^{-1} z_{t+1}$$

$$\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j+1} = U(I - \beta A)^{-1} A z_t$$

Using these formulas together with (7.3) and substituting into (7.16) and (7.17) gives the following representation for the consumer's optimum decision rule:

$$\begin{aligned} c_{t+1} &= c_t + (1-\beta) U (I - \beta A)^{-1} C w_{t+1} \\ b_t &= U (I - \beta A)^{-1} z_t - \frac{1}{1-\beta} c_t \\ y_t &= U z_t \\ z_{t+1} &= A z_t + C w_{t+1} \end{aligned} \tag{7.18}$$

Representation (7.18) makes clear that

- The state can be taken as (c_t, z_t).
  - The endogenous part is c_t and the exogenous part is z_t.
  - Debt b_t has disappeared as a component of the state because it is encoded in c_t.
- Consumption is a random walk with innovation $(1-\beta) U (I - \beta A)^{-1} C w_{t+1}$.
  - This is a more explicit representation of the martingale result in (7.6).

7.3.2 Cointegration

Representation (7.18) reveals that the joint process $\{c_t, b_t\}$ possesses the property that Engle and Granger [Engle and Granger, 1987] called cointegration.

Cointegration is a tool that allows us to apply powerful results from the theory of stationary stochastic processes to (certain transformations of) nonstationary models.

To apply cointegration in the present context, suppose that z_t is asymptotically stationary[3].

Despite this, both c_t and b_t will be non-stationary because they have unit roots (see (7.11) for b_t).

Nevertheless, there is a linear combination of c_t, b_t that *is* asymptotically stationary.

In particular, from the second equality in (7.18) we have

$$(1-\beta) b_t + c_t = (1-\beta) U (I - \beta A)^{-1} z_t \tag{7.19}$$

Hence the linear combination $(1-\beta) b_t + c_t$ is asymptotically stationary.

Accordingly, Granger and Engle would call $\begin{bmatrix} (1-\beta) & 1 \end{bmatrix}$ a **cointegrating vector** for the state.

When applied to the nonstationary vector process $\begin{bmatrix} b_t & c_t \end{bmatrix}'$, it yields a process that is asymptotically stationary.

[3] This would be the case if, for example, the spectral radius of A is strictly less than one.

Equation (7.19) can be rearranged to take the form

$$(1-\beta)b_t + c_t = (1-\beta)\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} \tag{7.20}$$

Equation (7.20) asserts that the *cointegrating residual* on the left side equals the conditional expectation of the geometric sum of future incomes on the right[4].

7.3.3 Cross-Sectional Implications

Consider again (7.18), this time in light of our discussion of distribution dynamics in the *lecture on linear systems*.

The dynamics of c_t are given by

$$c_{t+1} = c_t + (1-\beta)U(I-\beta A)^{-1}Cw_{t+1} \tag{7.21}$$

or

$$c_t = c_0 + \sum_{j=1}^{t} \hat{w}_j \quad \text{for} \quad \hat{w}_{t+1} := (1-\beta)U(I-\beta A)^{-1}Cw_{t+1}$$

The unit root affecting c_t causes the time t variance of c_t to grow linearly with t.

In particular, since $\{\hat{w}_t\}$ is IID, we have

$$\text{Var}[c_t] = \text{Var}[c_0] + t\,\hat{\sigma}^2 \tag{7.22}$$

where

$$\hat{\sigma}^2 := (1-\beta)^2 U(I-\beta A)^{-1}CC'(I-\beta A')^{-1}U'$$

When $\hat{\sigma} > 0$, $\{c_t\}$ has no asymptotic distribution.

Let's consider what this means for a cross-section of ex-ante identical consumers born at time 0.

Let the distribution of c_0 represent the cross-section of initial consumption values.

Equation (7.22) tells us that the variance of c_t increases over time at a rate proportional to t.

A number of different studies have investigated this prediction and found some support for it (see, e.g., [Deaton and Paxson, 1994], [Storesletten *et al.*, 2004]).

7.3.4 Impulse Response Functions

Impulse response functions measure responses to various impulses (i.e., temporary shocks).

The impulse response function of $\{c_t\}$ to the innovation $\{w_t\}$ is a box.

In particular, the response of c_{t+j} to a unit increase in the innovation w_{t+1} is $(1-\beta)U(I-\beta A)^{-1}C$ for all $j \geq 1$.

[4] See [John Y. Campbell, 1988], [Lettau and Ludvigson, 2001], [Lettau and Ludvigson, 2004] for interesting applications of related ideas.

7.3.5 Moving Average Representation

It's useful to express the innovation to the expected present value of the endowment process in terms of a moving average representation for income y_t.

The endowment process defined by (7.3) has the moving average representation

$$y_{t+1} = d(L) w_{t+1} \tag{7.23}$$

where

- $d(L) = \sum_{j=0}^{\infty} d_j L^j$ for some sequence d_j, where L is the lag operator[5]
- at time t, the consumer has an information set[6] $w^t = [w_t, w_{t-1}, ...]$

Notice that

$$y_{t+j} - \mathbb{E}_t[y_{t+j}] = d_0 w_{t+j} + d_1 w_{t+j-1} + \cdots + d_{j-1} w_{t+1}$$

It follows that

$$\mathbb{E}_{t+1}[y_{t+j}] - \mathbb{E}_t[y_{t+j}] = d_{j-1} w_{t+1} \tag{7.24}$$

Using (7.24) in (7.16) gives

$$c_{t+1} - c_t = (1-\beta) d(\beta) w_{t+1} \tag{7.25}$$

The object $d(\beta)$ is the **present value of the moving average coefficients** in the representation for the endowment process y_t.

7.4 Two Classic Examples

We illustrate some of the preceding ideas with two examples.

In both examples, the endowment follows the process $y_t = z_{1t} + z_{2t}$ where

$$\begin{bmatrix} z_{1t+1} \\ z_{2t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z_{1t} \\ z_{2t} \end{bmatrix} + \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} w_{1t+1} \\ w_{2t+1} \end{bmatrix}$$

Here

- w_{t+1} is an IID 2×1 process distributed as $N(0, I)$.
- z_{1t} is a permanent component of y_t.
- z_{2t} is a purely transitory component of y_t.

7.4.1 Example 1

Assume as before that the consumer observes the state z_t at time t.

In view of (7.18) we have

$$c_{t+1} - c_t = \sigma_1 w_{1t+1} + (1-\beta)\sigma_2 w_{2t+1} \tag{7.26}$$

Formula (7.26) shows how an increment $\sigma_1 w_{1t+1}$ to the permanent component of income z_{1t+1} leads to

[5] Representation (7.3) implies that $d(L) = U(I - AL)^{-1} C$.

[6] A moving average representation for a process y_t is said to be **fundamental** if the linear space spanned by y^t is equal to the linear space spanned by w^t. A time-invariant innovations representation, attained via the Kalman filter, is by construction fundamental.

- a permanent one-for-one increase in consumption and
- no increase in savings $-b_{t+1}$

But the purely transitory component of income $\sigma_2 w_{2t+1}$ leads to a permanent increment in consumption by a fraction $1-\beta$ of transitory income.

The remaining fraction β is saved, leading to a permanent increment in $-b_{t+1}$.

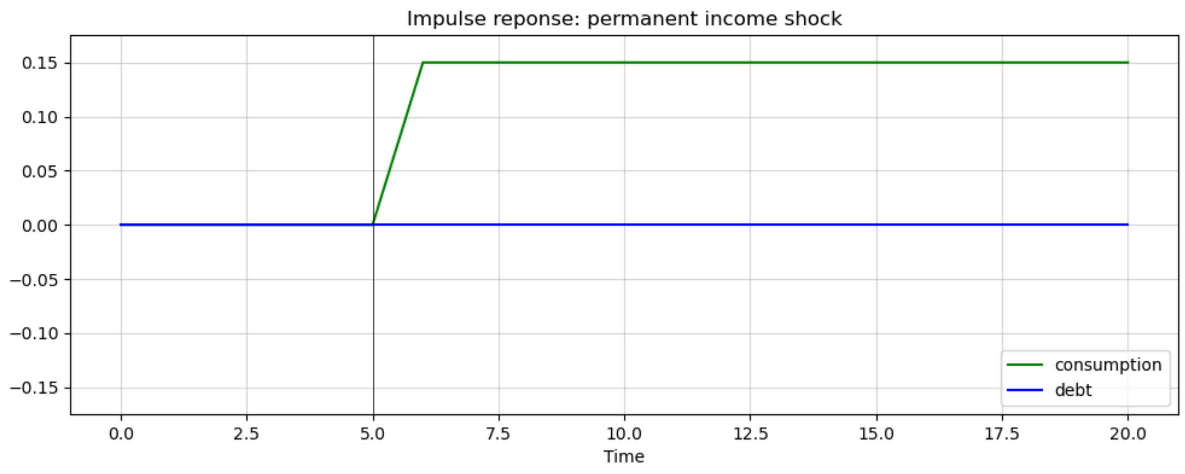
Application of the formula for debt in (7.11) to this example shows that

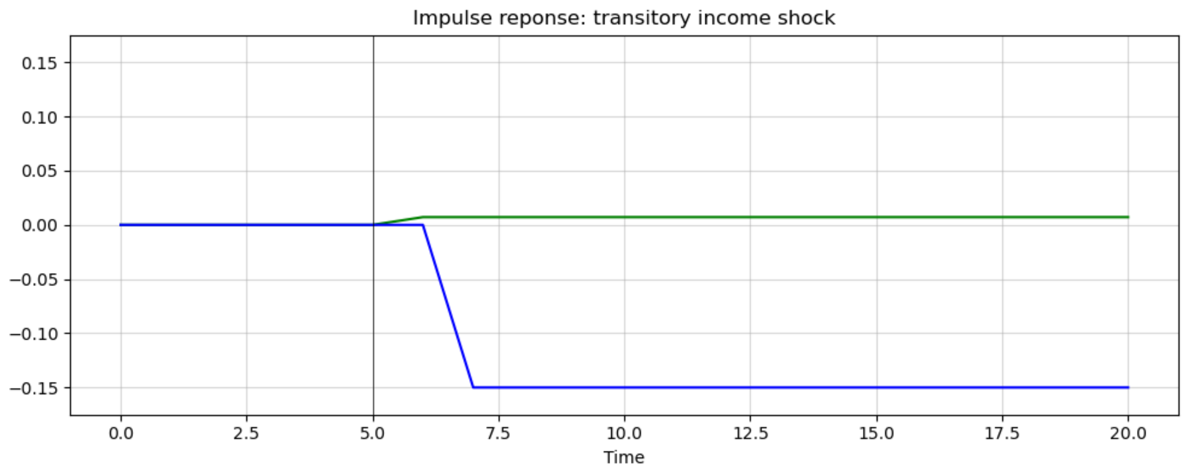
$$b_{t+1} - b_t = -z_{2t} = -\sigma_2 w_{2t} \tag{7.27}$$

This confirms that none of $\sigma_1 w_{1t}$ is saved, while all of $\sigma_2 w_{2t}$ is saved.

The next figure displays impulse-response functions that illustrates these very different reactions to transitory and permanent income shocks.

```python
r = 0.05
β = 1 / (1 + r)
S = 5   # Impulse date
σ1 = σ2 = 0.15

@njit
def time_path(T, permanent=False):
    "Time path of consumption and debt given shock sequence"
    w1 = np.zeros(T+1)
    w2 = np.zeros(T+1)
    b = np.zeros(T+1)
    c = np.zeros(T+1)
    if permanent:
        w1[S+1] = 1.0
    else:
        w2[S+1] = 1.0
    for t in range(1, T):
        b[t+1] = b[t] - σ2 * w2[t]
        c[t+1] = c[t] + σ1 * w1[t+1] + (1 - β) * σ2 * w2[t+1]
    return b, c


fig, axes = plt.subplots(2, 1, figsize=(10, 8))
titles = ['permanent', 'transitory']

L = 0.175

for ax, truefalse, title in zip(axes, (True, False), titles):
    b, c = time_path(T=20, permanent=truefalse)
    ax.set_title(f'Impulse reponse: {title} income shock')
    ax.plot(c, 'g-', label="consumption")
    ax.plot(b, 'b-', label="debt")
    ax.plot((S, S), (-L, L), 'k-', lw=0.5)
    ax.grid(alpha=0.5)
    ax.set(xlabel=r'Time', ylim=(-L, L))

axes[0].legend(loc='lower right')

plt.tight_layout()
plt.show()
```





Notice how the permanent income shock provokes no change in assets $-b_{t+1}$ and an immediate permanent change in consumption equal to the permanent increment in non-financial income.

In contrast, notice how most of a transitory income shock is saved and only a small amount is saved.

The box-like impulse responses of consumption to both types of shock reflect the random walk property of the optimal consumption decision.

7.4.2 Example 2

Assume now that at time t the consumer observes y_t, and its history up to t, but not z_t.

Under this assumption, it is appropriate to use an *innovation representation* to form A, C, U in (7.18).

The discussion in sections 2.9.1 and 2.11.3 of [Ljungqvist and Sargent, 2018] shows that the pertinent state space representation for y_t is

$$\begin{bmatrix} y_{t+1} \\ a_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & -(1-K) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ a_t \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} a_{t+1}$$

$$y_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ a_t \end{bmatrix}$$

where

- $K :=$ the stationary Kalman gain

- $a_t := y_t - E[y_t \mid y_{t-1}, \ldots, y_0]$

In the same discussion in [Ljungqvist and Sargent, 2018] it is shown that $K \in [0, 1]$ and that K increases as σ_1/σ_2 does.

In other words, K increases as the ratio of the standard deviation of the permanent shock to that of the transitory shock increases.

Please see *first look at the Kalman filter*.

Applying formulas (7.18) implies

$$c_{t+1} - c_t = [1 - \beta(1 - K)]a_{t+1} \tag{7.28}$$

where the endowment process can now be represented in terms of the univariate innovation to y_t as

$$y_{t+1} - y_t = a_{t+1} - (1 - K)a_t \tag{7.29}$$

Equation (7.29) indicates that the consumer regards

- fraction K of an innovation a_{t+1} to y_{t+1} as *permanent*
- fraction $1 - K$ as purely transitory

The consumer permanently increases his consumption by the full amount of his estimate of the permanent part of a_{t+1}, but by only $(1 - \beta)$ times his estimate of the purely transitory part of a_{t+1}.

Therefore, in total, he permanently increments his consumption by a fraction $K + (1 - \beta)(1 - K) = 1 - \beta(1 - K)$ of a_{t+1}.

He saves the remaining fraction $\beta(1 - K)$.

According to equation (7.29), the first difference of income is a first-order moving average.

Equation (7.28) asserts that the first difference of consumption is IID.

Application of formula to this example shows that

$$b_{t+1} - b_t = (K - 1)a_t \tag{7.30}$$

This indicates how the fraction K of the innovation to y_t that is regarded as permanent influences the fraction of the innovation that is saved.

7.5 Further Reading

The model described above significantly changed how economists think about consumption.

While Hall's model does a remarkably good job as a first approximation to consumption data, it's widely believed that it doesn't capture important aspects of some consumption/savings data.

For example, liquidity constraints and precautionary savings appear to be present sometimes.

Further discussion can be found in, e.g., [Hall and Mishkin, 1982], [Parker, 1999], [Deaton, 1991], [Carroll, 2001].

7.6 Appendix: The Euler Equation

Where does the first-order condition (7.5) come from?

Here we'll give a proof for the two-period case, which is representative of the general argument.

The finite horizon equivalent of the no-Ponzi condition is that the agent cannot end her life in debt, so $b_2 = 0$.

From the budget constraint (7.2) we then have

$$c_0 = \frac{b_1}{1+r} - b_0 + y_0 \quad \text{and} \quad c_1 = y_1 - b_1$$

Here b_0 and y_0 are given constants.

Substituting these constraints into our two-period objective $u(c_0) + \beta \mathbb{E}_0[u(c_1)]$ gives

$$\max_{b_1} \left\{ u\left(\frac{b_1}{R} - b_0 + y_0\right) + \beta \, \mathbb{E}_0[u(y_1 - b_1)] \right\}$$

You will be able to verify that the first-order condition is

$$u'(c_0) = \beta R \, \mathbb{E}_0[u'(c_1)]$$

Using $\beta R = 1$ gives (7.5) in the two-period case.

The proof for the general case is similar.

CHAPTER

EIGHT

PERMANENT INCOME II: LQ TECHNIQUES

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install quantecon
```

8.1 Overview

This lecture continues our analysis of the linear-quadratic (LQ) permanent income model of savings and consumption.

As we saw in our *previous lecture* on this topic, Robert Hall [Hall, 1978] used the LQ permanent income model to restrict and interpret intertemporal comovements of nondurable consumption, nonfinancial income, and financial wealth.

For example, we saw how the model asserts that for any covariance stationary process for nonfinancial income

- consumption is a random walk
- financial wealth has a unit root and is cointegrated with consumption

Other applications use the same LQ framework.

For example, a model isomorphic to the LQ permanent income model has been used by Robert Barro [Barro, 1979] to interpret intertemporal comovements of a government's tax collections, its expenditures net of debt service, and its public debt.

This isomorphism means that in analyzing the LQ permanent income model, we are in effect also analyzing the Barro tax smoothing model.

It is just a matter of appropriately relabeling the variables in Hall's model.

In this lecture, we'll

- show how the solution to the LQ permanent income model can be obtained using LQ control methods.

- represent the model as a linear state space system as in *this lecture*.
- apply QuantEcon's LinearStateSpace class to characterize statistical features of the consumer's optimal consumption and borrowing plans.

We'll then use these characterizations to construct a simple model of cross-section wealth and consumption dynamics in the spirit of Truman Bewley [Bewley, 1986].

(Later we'll study other Bewley models—see this lecture.)

The model will prove useful for illustrating concepts such as

- stationarity
- ergodicity
- ensemble moments and cross-section observations

Let's start with some imports:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import quantecon as qe
import numpy as np
import scipy.linalg as la
```

8.2 Setup

Let's recall the basic features of the model discussed in the *permanent income model*.

Consumer preferences are ordered by

$$E_0 \sum_{t=0}^{\infty} \beta^t u(c_t) \tag{8.1}$$

where $u(c) = -(c-\gamma)^2$.

The consumer maximizes (8.1) by choosing a consumption, borrowing plan $\{c_t, b_{t+1}\}_{t=0}^{\infty}$ subject to the sequence of budget constraints

$$c_t + b_t = \frac{1}{1+r} b_{t+1} + y_t, \quad t \geq 0 \tag{8.2}$$

and the no-Ponzi condition

$$E_0 \sum_{t=0}^{\infty} \beta^t b_t^2 < \infty \tag{8.3}$$

The interpretation of all variables and parameters are the same as in the *previous lecture*.

We continue to assume that $(1+r)\beta = 1$.

The dynamics of $\{y_t\}$ again follow the linear state space model

$$\begin{aligned} z_{t+1} &= A z_t + C w_{t+1} \\ y_t &= U z_t \end{aligned} \tag{8.4}$$

The restrictions on the shock process and parameters are the same as in our *previous lecture*.

8.2.1 Digression on a Useful Isomorphism

The LQ permanent income model of consumption is mathematically isomorphic with a version of Barro's [Barro, 1979] model of tax smoothing.

In the LQ permanent income model

- the household faces an exogenous process of nonfinancial income
- the household wants to smooth consumption across states and time

In the Barro tax smoothing model

- a government faces an exogenous sequence of government purchases (net of interest payments on its debt)
- a government wants to smooth tax collections across states and time

If we set

- T_t, total tax collections in Barro's model to consumption c_t in the LQ permanent income model.
- G_t, exogenous government expenditures in Barro's model to nonfinancial income y_t in the permanent income model.
- B_t, government risk-free one-period assets falling due in Barro's model to risk-free one-period consumer debt b_t falling due in the LQ permanent income model.
- R, the gross rate of return on risk-free one-period government debt in Barro's model to the gross rate of return $1 + r$ on financial assets in the permanent income model of consumption.

then the two models are mathematically equivalent.

All characterizations of a $\{c_t, y_t, b_t\}$ in the LQ permanent income model automatically apply to a $\{T_t, G_t, B_t\}$ process in the Barro model of tax smoothing.

See consumption and tax smoothing models for further exploitation of an isomorphism between consumption and tax smoothing models.

8.2.2 A Specification of the Nonfinancial Income Process

For the purposes of this lecture, let's assume $\{y_t\}$ is a second-order univariate autoregressive process:

$$y_{t+1} = \alpha + \rho_1 y_t + \rho_2 y_{t-1} + \sigma w_{t+1}$$

We can map this into the linear state space framework in (8.4), as discussed in our lecture on *linear models*.

To do so we take

$$z_t = \begin{bmatrix} 1 \\ y_t \\ y_{t-1} \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & 0 \\ \alpha & \rho_1 & \rho_2 \\ 0 & 1 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 \\ \sigma \\ 0 \end{bmatrix}, \quad \text{and} \quad U = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

8.3 The LQ Approach

Previously we solved the permanent income model by solving a system of linear expectational difference equations subject to two boundary conditions.

Here we solve the same model using *LQ methods* based on dynamic programming.

After confirming that answers produced by the two methods agree, we apply QuantEcon's LinearStateSpace class to illustrate features of the model.

Why solve a model in two distinct ways?

Because by doing so we gather insights about the structure of the model.

Our earlier approach based on solving a system of expectational difference equations brought to the fore the role of the consumer's expectations about future nonfinancial income.

On the other hand, formulating the model in terms of an LQ dynamic programming problem reminds us that

- finding the state (of a dynamic programming problem) is an art, and
- iterations on a Bellman equation implicitly jointly solve both a forecasting problem and a control problem

8.3.1 The LQ Problem

Recall from our *lecture on LQ theory* that the optimal linear regulator problem is to choose a decision rule for u_t to minimize

$$\mathbb{E}\sum_{t=0}^{\infty}\beta^t\{x_t'Rx_t+u_t'Qu_t\},$$

subject to x_0 given and the law of motion

$$x_{t+1}=\tilde{A}x_t+\tilde{B}u_t+\tilde{C}w_{t+1}, \qquad t\geq 0, \tag{8.5}$$

where w_{t+1} is IID with mean vector zero and $\mathbb{E}w_tw_t'=I$.

The tildes in $\tilde{A},\tilde{B},\tilde{C}$ are to avoid clashing with notation in (8.4).

The value function for this problem is $v(x)=-x'Px-d$, where

- P is the unique positive semidefinite solution of the *corresponding matrix Riccati equation*.
- The scalar d is given by $d=\beta(1-\beta)^{-1}\text{trace}(P\tilde{C}\tilde{C}')$.

The optimal policy is $u_t=-Fx_t$, where $F:=\beta(Q+\beta\tilde{B}'P\tilde{B})^{-1}\tilde{B}'P\tilde{A}$.

Under an optimal decision rule F, the state vector x_t evolves according to $x_{t+1}=(\tilde{A}-\tilde{B}F)x_t+\tilde{C}w_{t+1}$.

8.3.2 Mapping into the LQ Framework

To map into the LQ framework, we'll use

$$x_t:=\begin{bmatrix}z_t\\b_t\end{bmatrix}=\begin{bmatrix}1\\y_t\\y_{t-1}\\b_t\end{bmatrix}$$

as the state vector and $u_t:=c_t-\gamma$ as the control.

With this notation and $U_\gamma := [\gamma \quad 0 \quad 0]$, we can write the state dynamics as in (8.5) when

$$\tilde{A} := \begin{bmatrix} A & 0 \\ (1+r)(U_\gamma - U) & 1+r \end{bmatrix} \quad \tilde{B} := \begin{bmatrix} 0 \\ 1+r \end{bmatrix} \quad \text{and} \quad \tilde{C} := \begin{bmatrix} C \\ 0 \end{bmatrix} w_{t+1}$$

Please confirm for yourself that, with these definitions, the LQ dynamics (8.5) match the dynamics of z_t and b_t described above.

To map utility into the quadratic form $x_t' R x_t + u_t' Q u_t$ we can set

- $Q := 1$ (remember that we are minimizing) and
- $R :=$ a 4×4 matrix of zeros

However, there is one problem remaining.

We have no direct way to capture the non-recursive restriction (8.3) on the debt sequence $\{b_t\}$ from within the LQ framework.

To try to enforce it, we're going to use a trick: put a small penalty on b_t^2 in the criterion function.

In the present setting, this means adding a small entry $\epsilon > 0$ in the $(4, 4)$ position of R.

That will induce a (hopefully) small approximation error in the decision rule.

We'll check whether it really is small numerically soon.

8.4 Implementation

Let's write some code to solve the model.

One comment before we start is that the bliss level of consumption γ in the utility function has no effect on the optimal decision rule.

We saw this in the previous lecture *permanent income*.

The reason is that it drops out of the Euler equation for consumption.

In what follows we set it equal to unity.

8.4.1 The Exogenous Nonfinancial Income Process

First, we create the objects for the optimal linear regulator

```
# Set parameters
α, β, ρ1, ρ2, σ = 10.0, 0.95, 0.9, 0.0, 1.0

R = 1 / β
A = np.array([[1., 0., 0.],
              [α,  ρ1, ρ2],
              [0., 1., 0.]])
C = np.array([[0.], [σ], [0.]])
G = np.array([[0., 1., 0.]])

# Form LinearStateSpace system and pull off steady state moments
μ_z0 = np.array([[1.0], [0.0], [0.0]])
Σ_z0 = np.zeros((3, 3))
Lz = qe.LinearStateSpace(A, C, G, mu_0=μ_z0, Sigma_0=Σ_z0)
```

(continues on next page)

(continued from previous page)

```
μ_z, μ_y, Σ_z, Σ_y, Σ_yx = Lz.stationary_distributions()

# Mean vector of state for the savings problem
mxo = np.vstack([μ_z, 0.0])

# Create stationary covariance matrix of x -- start everyone off at b=0
a1 = np.zeros((3, 1))
aa = np.hstack([Σ_z, a1])
bb = np.zeros((1, 4))
sxo = np.vstack([aa, bb])

# These choices will initialize the state vector of an individual at zero
# debt and the ergodic distribution of the endowment process. Use these to
# create the Bewley economy.
mxbewley = mxo
sxbewley = sxo
```

The next step is to create the matrices for the LQ system

```
A12 = np.zeros((3,1))
ALQ_l = np.hstack([A, A12])
ALQ_r = np.array([[0, -R, 0, R]])
ALQ = np.vstack([ALQ_l, ALQ_r])

RLQ = np.array([[0., 0., 0., 0.],
                [0., 0., 0., 0.],
                [0., 0., 0., 0.],
                [0., 0., 0., 1e-9]])

QLQ = np.array([1.0])
BLQ = np.array([0., 0., 0., R]).reshape(4,1)
CLQ = np.array([0., σ, 0., 0.]).reshape(4,1)
β_LQ = β
```

Let's print these out and have a look at them

```
print(f"A = \n {ALQ}")
print(f"B = \n {BLQ}")
print(f"R = \n {RLQ}")
print(f"Q = \n {QLQ}")
```

```
A =
 [[ 1.          0.          0.          0.        ]
 [10.          0.9         0.          0.        ]
 [ 0.          1.          0.          0.        ]
 [ 0.         -1.05263158  0.          1.05263158]]
B =
 [[0.        ]
 [0.        ]
 [0.        ]
 [1.05263158]]
R =
 [[0.e+00 0.e+00 0.e+00 0.e+00]
 [0.e+00 0.e+00 0.e+00 0.e+00]
 [0.e+00 0.e+00 0.e+00 0.e+00]
```

(continues on next page)

(continued from previous page)

```
 [0.e+00 0.e+00 0.e+00 1.e-09]]
Q =
 [1.]
```

Now create the appropriate instance of an LQ model

```
lqpi = qe.LQ(QLQ, RLQ, ALQ, BLQ, C=CLQ, beta=β_LQ)
```

We'll save the implied optimal policy function soon compare them with what we get by employing an alternative solution method

```
P, F, d = lqpi.stationary_values()  # Compute value function and decision rule
ABF = ALQ - BLQ @ F  #  Form closed loop system
```

8.4.2 Comparison with the Difference Equation Approach

In our *first lecture* on the infinite horizon permanent income problem we used a different solution method.

The method was based around

- deducing the Euler equations that are the first-order conditions with respect to consumption and savings.
- using the budget constraints and boundary condition to complete a system of expectational linear difference equations.
- solving those equations to obtain the solution.

Expressed in state space notation, the solution took the form

$$\begin{aligned}
z_{t+1} &= Az_t + Cw_{t+1} \\
b_{t+1} &= b_t + U[(I-\beta A)^{-1}(A-I)]z_t \\
y_t &= Uz_t \\
c_t &= (1-\beta)[U(I-\beta A)^{-1}z_t - b_t]
\end{aligned}$$

Now we'll apply the formulas in this system

```
# Use the above formulas to create the optimal policies for b_{t+1} and c_t
b_pol = G @ la.inv(np.eye(3, 3) - β * A) @ (A - np.eye(3, 3))
c_pol = (1 - β) * G @ la.inv(np.eye(3, 3) - β * A)

# Create the A matrix for a LinearStateSpace instance
A_LSS1 = np.vstack([A, b_pol])
A_LSS2 = np.eye(4, 1, -3)
A_LSS = np.hstack([A_LSS1, A_LSS2])

# Create the C matrix for LSS methods
C_LSS = np.vstack([C, np.zeros(1)])

# Create the G matrix for LSS methods
G_LSS1 = np.vstack([G, c_pol])
G_LSS2 = np.vstack([np.zeros(1), -(1 - β)])
G_LSS = np.hstack([G_LSS1, G_LSS2])

# Use the following values to start everyone off at b=0, initial incomes zero
```

(continues on next page)

(continued from previous page)

```
μ_0 = np.array([1., 0., 0., 0.])
Σ_0 = np.zeros((4, 4))
```

A_LSS calculated as we have here should equal ABF calculated above using the LQ model

```
ABF - A_LSS
```

```
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [-9.51248181e-06,  9.51247878e-08,  0.00000000e+00,
        -1.99999901e-08]])
```

Now compare pertinent elements of c_pol and F

```
print(c_pol, "\n", -F)
```

```
[[65.51724138  0.34482759  0.        ]]
 [[ 6.55172323e+01  3.44827677e-01 -0.00000000e+00 -5.00000190e-02]]
```

We have verified that the two methods give the same solution.

Now let's create instances of the LinearStateSpace class and use it to do some interesting experiments.

To do this, we'll use the outcomes from our second method.

8.5 Two Example Economies

In the spirit of Bewley models [Bewley, 1986], we'll generate panels of consumers.

The examples differ only in the initial states with which we endow the consumers.

All other parameter values are kept the same in the two examples

- In the first example, all consumers begin with zero nonfinancial income and zero debt.
  - The consumers are thus *ex-ante* identical.
- In the second example, while all begin with zero debt, we draw their initial income levels from the invariant distribution of financial income.
  - Consumers are *ex-ante* heterogeneous.

In the first example, consumers' nonfinancial income paths display pronounced transients early in the sample

- these will affect outcomes in striking ways

Those transient effects will not be present in the second example.

We use methods affiliated with the LinearStateSpace class to simulate the model.

8.5.1 First Set of Initial Conditions

We generate 25 paths of the exogenous non-financial income process and the associated optimal consumption and debt paths.

In the first set of graphs, darker lines depict a particular sample path, while the lighter lines describe 24 other paths.

A second graph plots a collection of simulations against the population distribution that we extract from the `LinearStateSpace` instance `LSS`.

Comparing sample paths with population distributions at each date t is a useful exercise—see our discussion of the laws of large numbers

```
lss = qe.LinearStateSpace(A_LSS, C_LSS, G_LSS, mu_0=μ_0, Sigma_0=Σ_0)
```

8.5.2 Population and Sample Panels

In the code below, we use the LinearStateSpace class to

- compute and plot population quantiles of the distributions of consumption and debt for a population of consumers.
- simulate a group of 25 consumers and plot sample paths on the same graph as the population distribution.

```
def income_consumption_debt_series(A, C, G, μ_0, Σ_0, T=150, npaths=25):
    """
    This function takes initial conditions (μ_0, Σ_0) and uses the
    LinearStateSpace class from QuantEcon to  simulate an economy
    npaths times for T periods. It then uses that information to
    generate some graphs related to the discussion below.
    """
    lss = qe.LinearStateSpace(A, C, G, mu_0=μ_0, Sigma_0=Σ_0)

    # Simulation/Moment Parameters
    moment_generator = lss.moment_sequence()

    # Simulate various paths
    bsim = np.empty((npaths, T))
    csim = np.empty((npaths, T))
    ysim = np.empty((npaths, T))

    for i in range(npaths):
        sims = lss.simulate(T)
        bsim[i, :] = sims[0][-1, :]
        csim[i, :] = sims[1][1, :]
        ysim[i, :] = sims[1][0, :]

    # Get the moments
    cons_mean = np.empty(T)
    cons_var = np.empty(T)
    debt_mean = np.empty(T)
    debt_var = np.empty(T)
    for t in range(T):
        μ_x, μ_y, Σ_x, Σ_y = next(moment_generator)
        cons_mean[t], cons_var[t] = μ_y[1], Σ_y[1, 1]
        debt_mean[t], debt_var[t] = μ_x[3], Σ_x[3, 3]

    return bsim, csim, ysim, cons_mean, cons_var, debt_mean, debt_var
```

(continues on next page)

(continued from previous page)

```
def consumption_income_debt_figure(bsim, csim, ysim):

    # Get T
    T =  bsim.shape[1]

    # Create the first figure
    fig, ax = plt.subplots(2, 1, figsize=(10, 8))
    xvals = np.arange(T)

    # Plot consumption and income
    ax[0].plot(csim[0, :], label="c", color="b")
    ax[0].plot(ysim[0, :], label="y", color="g")
    ax[0].plot(csim.T, alpha=.1, color="b")
    ax[0].plot(ysim.T, alpha=.1, color="g")
    ax[0].legend(loc=4)
    ax[0].set(title="Nonfinancial Income, Consumption, and Debt",
              xlabel="t", ylabel="y and c")

    # Plot debt
    ax[1].plot(bsim[0, :], label="b", color="r")
    ax[1].plot(bsim.T, alpha=.1, color="r")
    ax[1].legend(loc=4)
    ax[1].set(xlabel="t", ylabel="debt")

    fig.tight_layout()
    return fig

def consumption_debt_fanchart(csim, cons_mean, cons_var,
                              bsim, debt_mean, debt_var):
    # Get T
    T =  bsim.shape[1]

    # Create percentiles of cross-section distributions
    cmean = np.mean(cons_mean)
    c90 = 1.65 * np.sqrt(cons_var)
    c95 = 1.96 * np.sqrt(cons_var)
    c_perc_95p, c_perc_95m = cons_mean + c95, cons_mean - c95
    c_perc_90p, c_perc_90m = cons_mean + c90, cons_mean - c90

    # Create percentiles of cross-section distributions
    dmean = np.mean(debt_mean)
    d90 = 1.65 * np.sqrt(debt_var)
    d95 = 1.96 * np.sqrt(debt_var)
    d_perc_95p, d_perc_95m = debt_mean + d95, debt_mean - d95
    d_perc_90p, d_perc_90m = debt_mean + d90, debt_mean - d90


    # Create second figure
    fig, ax = plt.subplots(2, 1, figsize=(10, 8))
    xvals = np.arange(T)

    # Consumption fan
    ax[0].plot(xvals, cons_mean, color="k")
    ax[0].plot(csim.T, color="k", alpha=.25)
    ax[0].fill_between(xvals, c_perc_95m, c_perc_95p, alpha=.25, color="b")
```

(continues on next page)

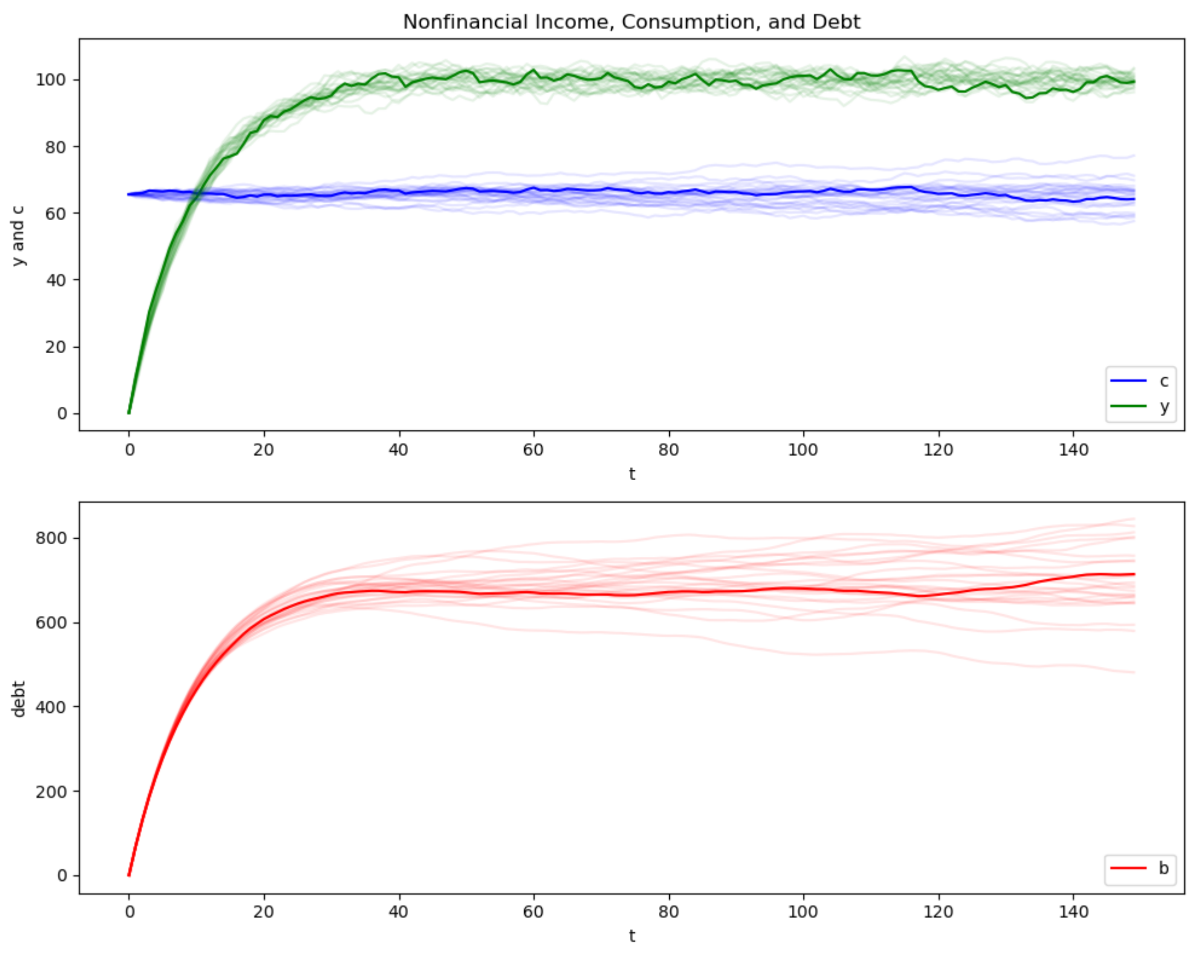
(continued from previous page)

```
    ax[0].fill_between(xvals, c_perc_90m, c_perc_90p, alpha=.25, color="r")
    ax[0].set(title="Consumption/Debt over time",
              ylim=(cmean-15, cmean+15), ylabel="consumption")

    # Debt fan
    ax[1].plot(xvals, debt_mean, color="k")
    ax[1].plot(bsim.T, color="k", alpha=.25)
    ax[1].fill_between(xvals, d_perc_95m, d_perc_95p, alpha=.25, color="b")
    ax[1].fill_between(xvals, d_perc_90m, d_perc_90p, alpha=.25, color="r")
    ax[1].set(xlabel="t", ylabel="debt")

    fig.tight_layout()
    return fig
```

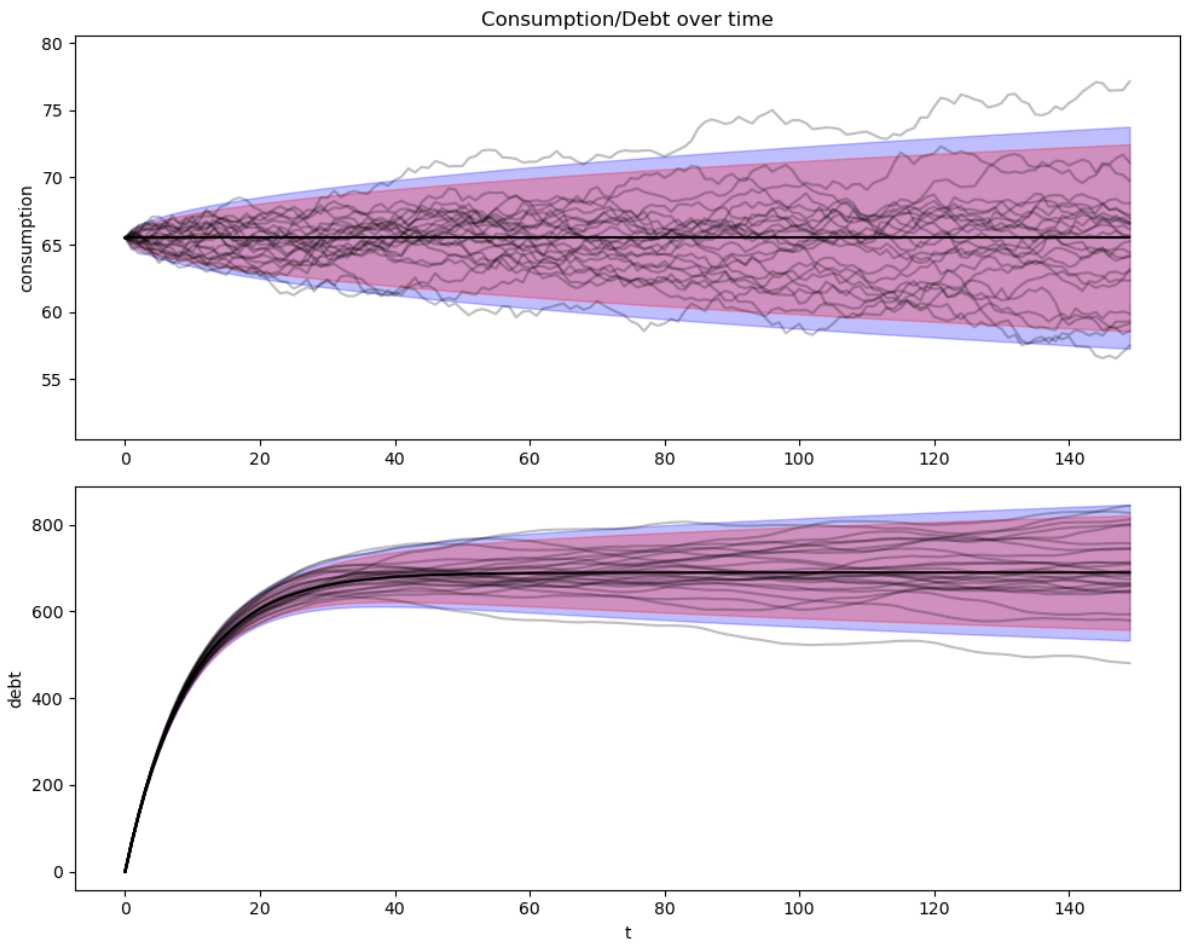
Now let's create figures with initial conditions of zero for y_0 and b_0

```
out = income_consumption_debt_series(A_LSS, C_LSS, G_LSS, μ_0, Σ_0)
bsim0, csim0, ysim0 = out[:3]
cons_mean0, cons_var0, debt_mean0, debt_var0 = out[3:]

consumption_income_debt_figure(bsim0, csim0, ysim0)

plt.show()
```

```
/tmp/ipykernel_6772/353008971.py:31: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  cons_mean[t], cons_var[t] = μ_y[1], Σ_y[1, 1]
/tmp/ipykernel_6772/353008971.py:32: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  debt_mean[t], debt_var[t] = μ_x[3], Σ_x[3, 3]
```



```
consumption_debt_fanchart(csim0, cons_mean0, cons_var0,
                          bsim0, debt_mean0, debt_var0)

plt.show()
```



Here is what is going on in the above graphs.

For our simulation, we have set initial conditions $b_0 = y_{-1} = y_{-2} = 0$.

Because $y_{-1} = y_{-2} = 0$, nonfinancial income y_t starts far below its stationary mean $\mu_{y,\infty}$ and rises early in each simulation.

Recall from the *previous lecture* that we can represent the optimal decision rule for consumption in terms of the **co-integrating relationship**

$$(1-\beta)b_t + c_t = (1-\beta)E_t \sum_{j=0}^{\infty} \beta^j y_{t+j} \tag{8.6}$$

So at time 0 we have

$$c_0 = (1-\beta)E_0 \sum_{t=0}^{\infty} \beta^j y_t$$

This tells us that consumption starts at the income that would be paid by an annuity whose value equals the expected discounted value of nonfinancial income at time $t = 0$.

To support that level of consumption, the consumer borrows a lot early and consequently builds up substantial debt.

In fact, he or she incurs so much debt that eventually, in the stochastic steady state, he consumes less each period than his nonfinancial income.

He uses the gap between consumption and nonfinancial income mostly to service the interest payments due on his debt.

Thus, when we look at the panel of debt in the accompanying graph, we see that this is a group of *ex-ante* identical people each of whom starts with zero debt.

All of them accumulate debt in anticipation of rising nonfinancial income.

They expect their nonfinancial income to rise toward the invariant distribution of income, a consequence of our having started them at $y_{-1} = y_{-2} = 0$.

Cointegration Residual

The following figure plots realizations of the left side of (8.6), which, *as discussed in our last lecture*, is called the **cointegrating residual**.

As mentioned above, the right side can be thought of as an annuity payment on the expected present value of future income $E_t \sum_{j=0}^{\infty} \beta^j y_{t+j}$.

Early along a realization, c_t is approximately constant while $(1-\beta)b_t$ and $(1-\beta)E_t \sum_{j=0}^{\infty} \beta^j y_{t+j}$ both rise markedly as the household's present value of income and borrowing rise pretty much together.

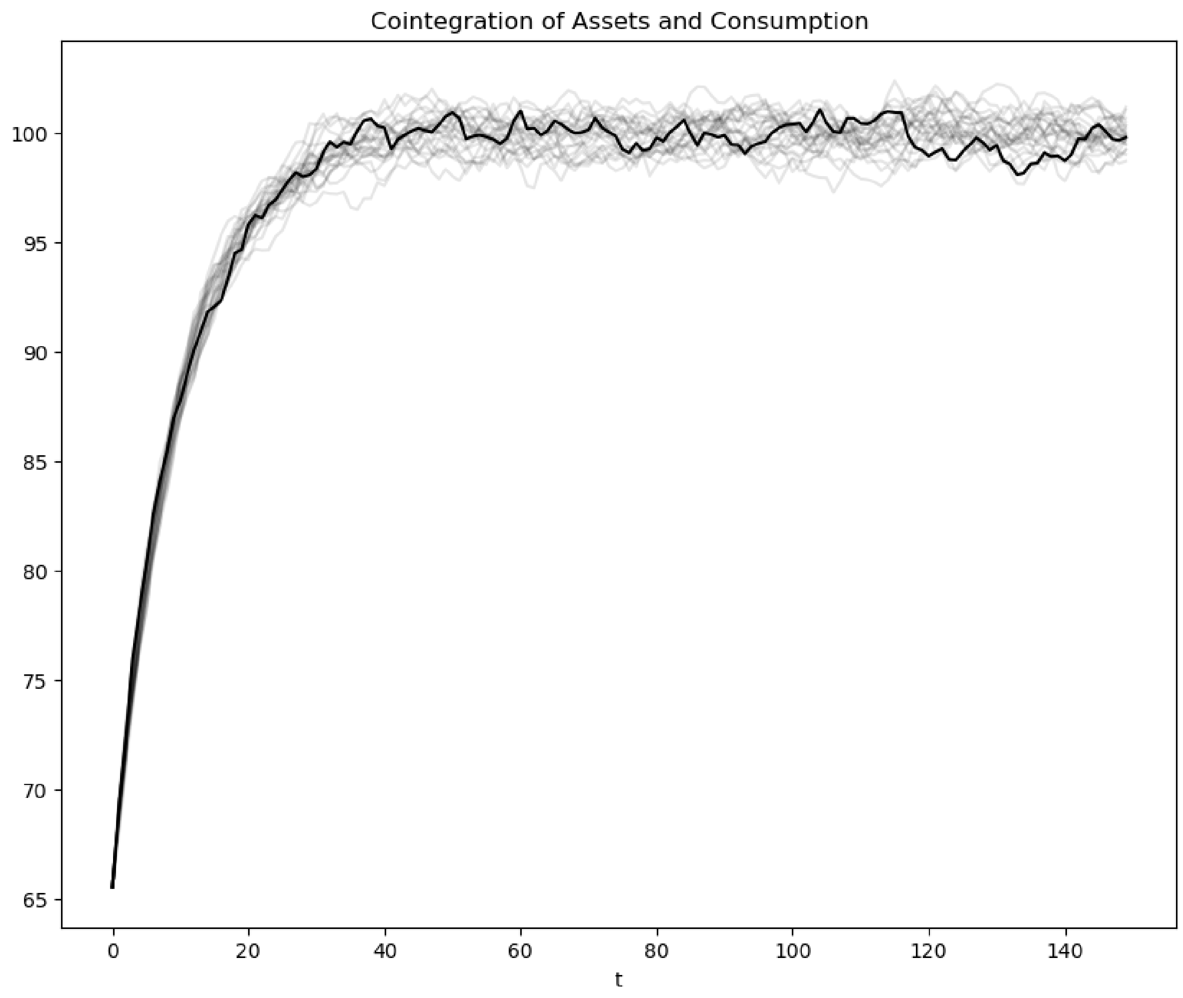
This example illustrates the following point: the definition of cointegration implies that the cointegrating residual is *asymptotically* covariance stationary, not *covariance stationary*.

The cointegrating residual for the specification with zero income and zero debt initially has a notable transient component that dominates its behavior early in the sample.

By altering initial conditions, we shall remove this transient in our second example to be presented below

```python
def cointegration_figure(bsim, csim):
    """
    Plots the cointegration
    """
    # Create figure
    fig, ax = plt.subplots(figsize=(10, 8))
    ax.plot((1 - β) * bsim[0, :] + csim[0, :], color="k")
    ax.plot((1 - β) * bsim.T + csim.T, color="k", alpha=.1)

    ax.set(title="Cointegration of Assets and Consumption", xlabel="t")

    return fig
```

```python
cointegration_figure(bsim0, csim0)
plt.show()
```



8.5.3 A "Borrowers and Lenders" Closed Economy

When we set $y_{-1} = y_{-2} = 0$ and $b_0 = 0$ in the preceding exercise, we make debt "head north" early in the sample.

Average debt in the cross-section rises and approaches the asymptote.

We can regard these as outcomes of a "small open economy" that borrows from abroad at the fixed gross interest rate $R = r + 1$ in anticipation of rising incomes.

So with the economic primitives set as above, the economy converges to a steady state in which there is an excess aggregate supply of risk-free loans at a gross interest rate of R.

This excess supply is filled by "foreigner lenders" willing to make those loans.

We can use virtually the same code to rig a "poor man's Bewley [Bewley, 1986] model" in the following way

- as before, we start everyone at $b_0 = 0$.
- But instead of starting everyone at $y_{-1} = y_{-2} = 0$, we draw $\begin{bmatrix} y_{-1} \\ y_{-2} \end{bmatrix}$ from the invariant distribution of the $\{y_t\}$ process.

This rigs a closed economy in which people are borrowing and lending with each other at a gross risk-free interest rate of $R = \beta^{-1}$.

Across the group of people being analyzed, risk-free loans are in zero excess supply.

We have arranged primitives so that $R = \beta^{-1}$ clears the market for risk-free loans at zero aggregate excess supply.

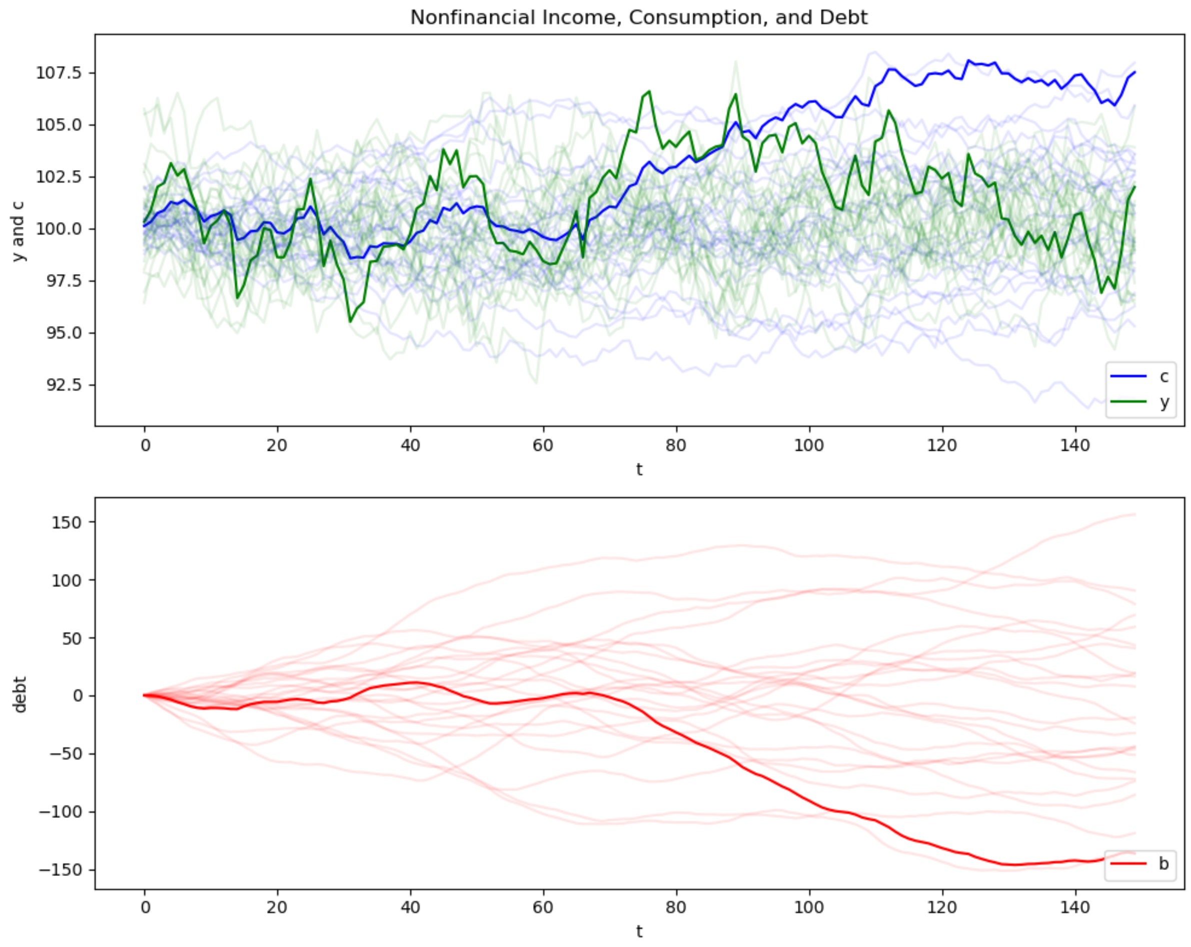
So the risk-free loans are being made from one person to another within our closed set of agents.

There is no need for foreigners to lend to our group.

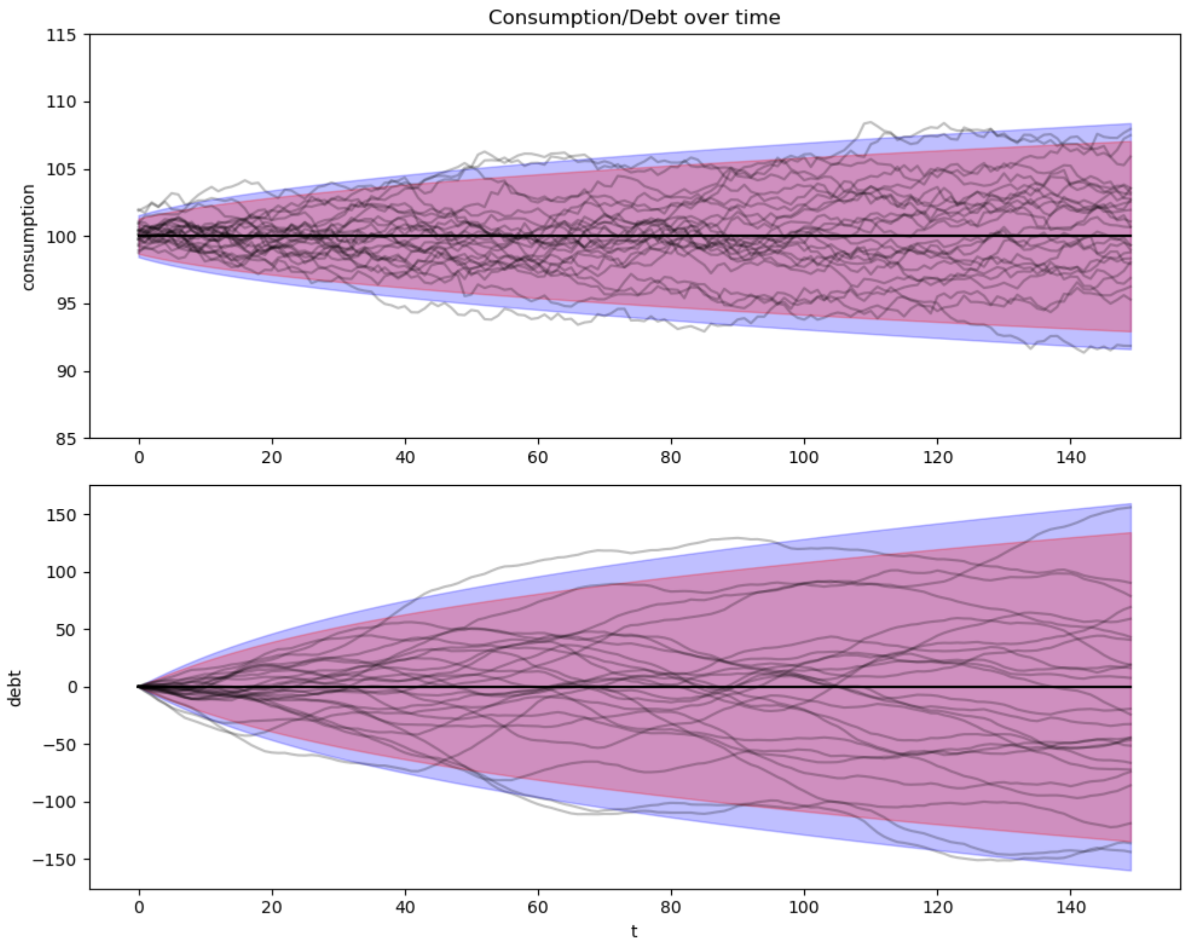
Let's have a look at the corresponding figures

```
out = income_consumption_debt_series(A_LSS, C_LSS, G_LSS, mxbewley, sxbewley)
bsimb, csimb, ysimb = out[:3]
cons_meanb, cons_varb, debt_meanb, debt_varb = out[3:]

consumption_income_debt_figure(bsimb, csimb, ysimb)

plt.show()
```

```
/tmp/ipykernel_6772/353008971.py:31: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  cons_mean[t], cons_var[t] = μ_y[1], Σ_y[1, 1]
/tmp/ipykernel_6772/353008971.py:32: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  debt_mean[t], debt_var[t] = μ_x[3], Σ_x[3, 3]
```



```
consumption_debt_fanchart(csimb, cons_meanb, cons_varb,
                          bsimb, debt_meanb, debt_varb)

plt.show()
```



The graphs confirm the following outcomes:

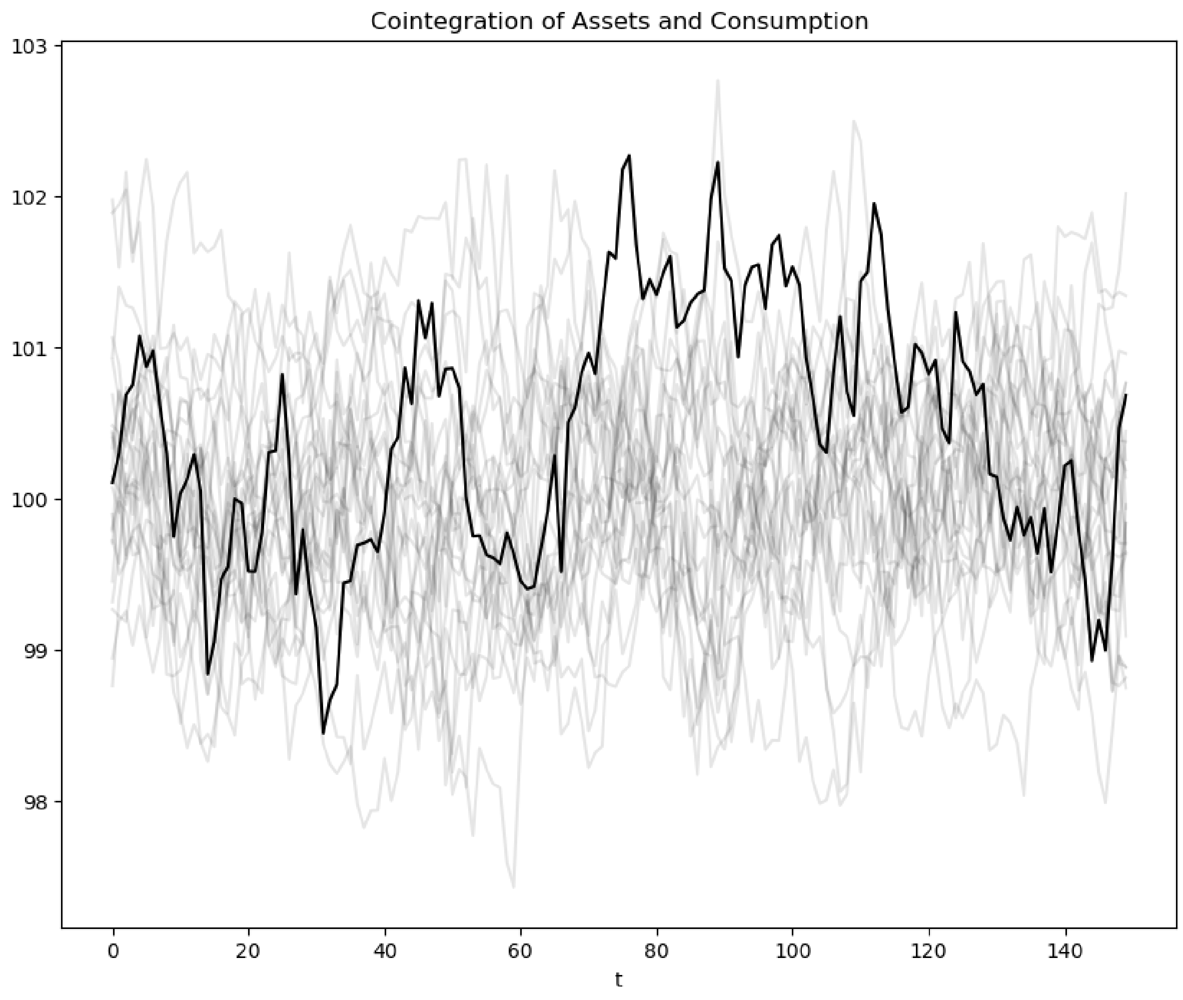
- As before, the consumption distribution spreads out over time.

But now there is some initial dispersion because there is *ex-ante* heterogeneity in the initial draws of $\begin{bmatrix} y_{-1} \\ y_{-2} \end{bmatrix}$.

- As before, the cross-section distribution of debt spreads out over time.
- Unlike before, the average level of debt stays at zero, confirming that this is a closed borrower-and-lender economy.
- Now the cointegrating residual seems stationary, and not just asymptotically stationary.

Let's have a look at the cointegration figure

```
cointegration_figure(bsimb, csimb)
plt.show()
```


Cointegration of Assets and Consumption
103
102
101
100
99
98
0
20
40
60
80
100
120
140
t

CHAPTER NINE

PRODUCTION SMOOTHING VIA INVENTORIES

Contents

In addition to what's in Anaconda, this lecture employs the following library:

```
!pip install quantecon
```

9.1 Overview

This lecture can be viewed as an application of this *quantecon lecture* about linear quadratic control theory.

It formulates a discounted dynamic program for a firm that chooses a production schedule to balance

- minimizing costs of production across time, against
- keeping costs of holding inventories low

In the tradition of a classic book by Holt, Modigliani, Muth, and Simon [Holt *et al.*, 1960], we simplify the firm's problem by formulating it as a linear quadratic discounted dynamic programming problem of the type studied in this *quantecon lecture*.

Because its costs of production are increasing and quadratic in production, the firm holds inventories as a buffer stock in order to smooth production across time, provided that holding inventories is not too costly.

But the firm also wants to make its sales out of existing inventories, a preference that we represent by a cost that is quadratic in the difference between sales in a period and the firm's beginning of period inventories.

We compute examples designed to indicate how the firm optimally smooths production while keeping inventories close to sales.

To introduce components of the model, let

- S_t be sales at time t
- Q_t be production at time t
- I_t be inventories at the beginning of time t
- $\beta \in (0, 1)$ be a discount factor
- $c(Q_t) = c_1 Q_t + c_2 Q_t^2$, be a cost of production function, where $c_1 > 0, c_2 > 0$, be an inventory cost function
- $d(I_t, S_t) = d_1 I_t + d_2 (S_t - I_t)^2$, where $d_1 > 0, d_2 > 0$, be a cost-of-holding-inventories function, consisting of two components:
  - a cost $d_1 I_t$ of carrying inventories, and
  - a cost $d_2 (S_t - I_t)^2$ of having inventories deviate from sales
- $p_t = a_0 - a_1 S_t + v_t$ be an inverse demand function for a firm's product, where $a_0 > 0, a_1 > 0$ and v_t is a demand shock at time t
- $\pi\_t = p_t S_t - c(Q_t) - d(I_t, S_t)$ be the firm's profits at time t
- $\sum_{t=0}^{\infty} \beta^t \pi_t$ be the present value of the firm's profits at time 0
- $I_{t+1} = I_t + Q_t - S_t$ be the law of motion of inventories
- $z_{t+1} = A_{22} z_t + C_2 \epsilon_{t+1}$ be a law of motion for an exogenous state vector z_t that contains time t information useful for predicting the demand shock v_t
- $v_t = G z_t$ link the demand shock to the information set z_t
- the constant 1 be the first component of z_t

To map our problem into a linear-quadratic discounted dynamic programming problem (also known as an optimal linear regulator), we define the **state** vector at time t as

$$x_t = \begin{bmatrix} I_t \\ z_t \end{bmatrix}$$

and the **control** vector as

$$u_t = \begin{bmatrix} Q_t \\ S_t \end{bmatrix}$$

The law of motion for the state vector x_t is evidently

$$\begin{bmatrix} I_{t+1} \\ z_t \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} I_t \\ z_t \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q_t \\ S_t \end{bmatrix} + \begin{bmatrix} 0 \\ C_2 \end{bmatrix} \epsilon_{t+1}$$

or

$$x_{t+1} = A x_t + B u_t + C \epsilon_{t+1}$$

(At this point, we ask that you please forgive us for using Q_t to be the firm's production at time t, while below we use Q as the matrix in the quadratic form $u_t' Q u_t$ that appears in the firm's one-period profit function)

We can express the firm's profit as a function of states and controls as

$$\pi_t = -(x_t' R x_t + u_t' Q u_t + 2 u_t' N x_t)$$

To form the matrices R, Q, N in an LQ dynamic programming problem, we note that the firm's profits at time t function can be expressed

$$
\begin{aligned}
\pi_t =& p_t S_t - c(Q_t) - d(I_t, S_t) \\
=& (a_0 - a_1 S_t + v_t) S_t - c_1 Q_t - c_2 Q_t^2 - d_1 I_t - d_2 (S_t - I_t)^2 \\
=& a_0 S_t - a_1 S_t^2 + G z_t S_t - c_1 Q_t - c_2 Q_t^2 - d_1 I_t - d_2 S_t^2 - d_2 I_t^2 + 2 d_2 S_t I_t \\
=& -\left(\underbrace{d_1 I_t + d_2 I_t^2}_{x_t' R x_t} \underbrace{+ a_1 S_t^2 + d_2 S_t^2 + c_2 Q_t^2}_{u_t' Q u_t} \underbrace{- a_0 S_t - G z_t S_t + c_1 Q_t - 2 d_2 S_t I_t}_{2 u_t' N x_t} \right) \\
=& -\left(\begin{bmatrix} I_t & z_t' \end{bmatrix} \underbrace{\begin{bmatrix} d_2 & \frac{d_1}{2} S_c \\ \frac{d_1}{2} S_c' & 0 \end{bmatrix}}_{\equiv R} \begin{bmatrix} I_t \\ z_t \end{bmatrix} + \begin{bmatrix} Q_t & S_t \end{bmatrix} \underbrace{\begin{bmatrix} c_2 & 0 \\ 0 & a_1 + d_2 \end{bmatrix}}_{\equiv Q} \begin{bmatrix} Q_t \\ S_t \end{bmatrix} + 2 \begin{bmatrix} Q_t & S_t \end{bmatrix} \underbrace{\begin{bmatrix} 0 & \frac{c_1}{2} S_c \\ -d_2 & -\frac{a_0}{2} S_c - \frac{G}{2} \end{bmatrix}}_{\equiv N} \left[\begin{matrix} I \\ z \end{matrix} \right.
\end{aligned}
$$

where $S_c = [1, 0]$.

Remark on notation: The notation for cross product term in the QuantEcon library is N.

The firms' optimum decision rule takes the form

$$u_t = -F x_t$$

and the evolution of the state under the optimal decision rule is

$$x_{t+1} = (A - BF) x_t + C \epsilon_{t+1}$$

The firm chooses a decision rule for u_t that maximizes

$$E_0 \sum_{t=0}^{\infty} \beta^t \pi_t$$

subject to a given x_0.

This is a stochastic discounted LQ dynamic program.

Here is code for computing an optimal decision rule and for analyzing its consequences.

```python
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
import quantecon as qe
```

```python
class SmoothingExample:
    """
    Class for constructing, solving, and plotting results for
    inventories and sales smoothing problem.
    """

    def __init__(self,
                 β=0.96,           # Discount factor
                 c1=1,             # Cost-of-production
                 c2=1,
                 d1=1,             # Cost-of-holding inventories
                 d2=1,
                 a0=10,            # Inverse demand function
```

(continues on next page)

(continued from previous page)

```
                 a1=1,
                 A22=[[1,   0],     # z process
                      [1, 0.9]],
                 C2=[[0], [1]],
                 G=[0, 1]):

        self.β = β
        self.c1, self.c2 = c1, c2
        self.d1, self.d2 = d1, d2
        self.a0, self.a1 = a0, a1
        self.A22 = np.atleast_2d(A22)
        self.C2 = np.atleast_2d(C2)
        self.G = np.atleast_2d(G)

        # Dimensions
        k, j = self.C2.shape        # Dimensions for randomness part
        n = k + 1                   # Number of states
        m = 2                       # Number of controls

        Sc = np.zeros(k)
        Sc[0] = 1

        # Construct matrices of transition law
        A = np.zeros((n, n))
        A[0, 0] = 1
        A[1:, 1:] = self.A22

        B = np.zeros((n, m))
        B[0, :] = 1, -1

        C = np.zeros((n, j))
        C[1:, :] = self.C2

        self.A, self.B, self.C = A, B, C

        # Construct matrices of one period profit function
        R = np.zeros((n, n))
        R[0, 0] = d2
        R[1:, 0] = d1 / 2 * Sc
        R[0, 1:] = d1 / 2 * Sc

        Q = np.zeros((m, m))
        Q[0, 0] = c2
        Q[1, 1] = a1 + d2

        N = np.zeros((m, n))
        N[1, 0] = - d2
        N[0, 1:] = c1 / 2 * Sc
        N[1, 1:] = - a0 / 2 * Sc - self.G / 2

        self.R, self.Q, self.N = R, Q, N

        # Construct LQ instance
        self.LQ = qe.LQ(Q, R, A, B, C, N, beta=β)
        self.LQ.stationary_values()
```

(continues on next page)

(continued from previous page)

```
    def simulate(self, x0, T=100):

        c1, c2 = self.c1, self.c2
        d1, d2 = self.d1, self.d2
        a0, a1 = self.a0, self.a1
        G = self.G

        x_path, u_path, w_path = self.LQ.compute_sequence(x0, ts_length=T)

        I_path = x_path[0, :-1]
        z_path = x_path[1:, :-1]
        �_path = (G @ z_path)[0, :]

        Q_path = u_path[0, :]
        S_path = u_path[1, :]

        revenue = (a0 - a1 * S_path + �_path) * S_path
        cost_production = c1 * Q_path + c2 * Q_path ** 2
        cost_inventories = d1 * I_path + d2 * (S_path - I_path) ** 2

        Q_no_inventory = (a0 + �_path - c1) / (2 * (a1 + c2))
        Q_hardwired = (a0 + �_path - c1) / (2 * (a1 + c2 + d2))

        fig, ax = plt.subplots(2, 2, figsize=(15, 10))

        ax[0, 0].plot(range(T), I_path, label="inventories")
        ax[0, 0].plot(range(T), S_path, label="sales")
        ax[0, 0].plot(range(T), Q_path, label="production")
        ax[0, 0].legend(loc=1)
        ax[0, 0].set_title("inventories, sales, and production")

        ax[0, 1].plot(range(T), (Q_path - S_path), color='b')
        ax[0, 1].set_ylabel("change in inventories", color='b')
        span = max(abs(Q_path - S_path))
        ax[0, 1].set_ylim(0-span*1.1, 0+span*1.1)
        ax[0, 1].set_title("demand shock and change in inventories")

        ax1_ = ax[0, 1].twinx()
        ax1_.plot(range(T), �_path, color='r')
        ax1_.set_ylabel("demand shock", color='r')
        span = max(abs(�_path))
        ax1_.set_ylim(0-span*1.1, 0+span*1.1)

        ax1_.plot([0, T], [0, 0], '--', color='k')

        ax[1, 0].plot(range(T), revenue, label="revenue")
        ax[1, 0].plot(range(T), cost_production, label="cost_production")
        ax[1, 0].plot(range(T), cost_inventories, label="cost_inventories")
        ax[1, 0].legend(loc=1)
        ax[1, 0].set_title("profits decomposition")

        ax[1, 1].plot(range(T), Q_path, label="production")
        ax[1, 1].plot(range(T), Q_hardwired, label='production when  $I_t$ \
            forced to be zero')
        ax[1, 1].plot(range(T), Q_no_inventory, label='production when \
            inventories not useful')
```

(continues on next page)

(continued from previous page)

```
        ax[1, 1].legend(loc=1)
        ax[1, 1].set_title('three production concepts')

        plt.show()
```

Notice that the above code sets parameters at the following default values

- discount factor $\beta = 0.96$,
- inverse demand function: $a0 = 10, a1 = 1$
- cost of production $c1 = 1, c2 = 1$
- costs of holding inventories $d1 = 1, d2 = 1$

In the examples below, we alter some or all of these parameter values.

9.2 Example 1

In this example, the demand shock follows AR(1) process:

$$\nu_t = \alpha + \rho \nu_{t-1} + \epsilon_t,$$

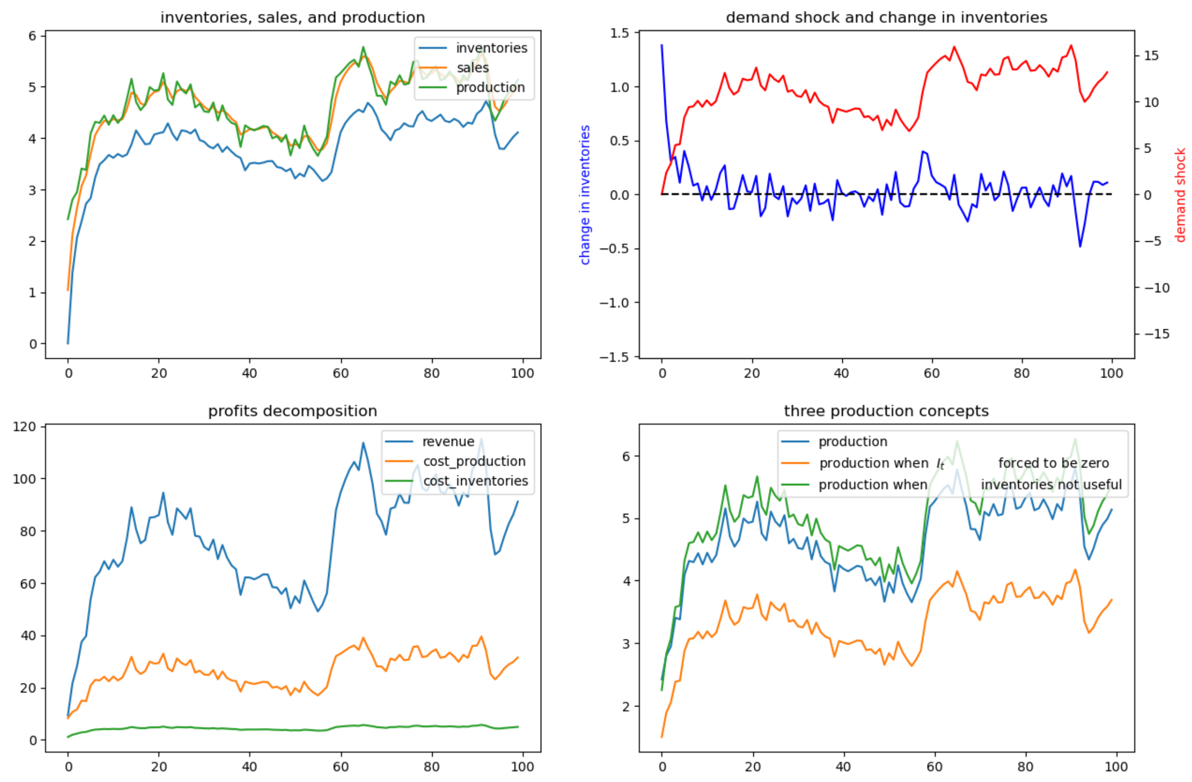
which implies

$$z_{t+1} = \begin{bmatrix} 1 \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \alpha & \rho \end{bmatrix} \underbrace{\begin{bmatrix} 1 \\ v_t \end{bmatrix}}_{z_t} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \epsilon_{t+1}.$$

We set $\alpha = 1$ and $\rho = 0.9$, their default values.

We'll calculate and display outcomes, then discuss them below the pertinent figures.

```
ex1 = SmoothingExample()

x0 = [0, 1, 0]
ex1.simulate(x0)
```



The figures above illustrate various features of an optimal production plan.

Starting from zero inventories, the firm builds up a stock of inventories and uses them to smooth costly production in the face of demand shocks.

Optimal decisions evidently respond to demand shocks.

Inventories are always less than sales, so some sales come from current production, a consequence of the cost, $d_1 I_t$ of holding inventories.

The lower right panel shows differences between optimal production and two alternative production concepts that come from altering the firm's cost structure – i.e., its technology.

These two concepts correspond to these distinct altered firm problems.

- a setting in which inventories are not needed
- a setting in which they are needed but we arbitrarily prevent the firm from holding inventories by forcing it to set $I_t = 0$ always

We use these two alternative production concepts in order to shed light on the baseline model.

9.3 Inventories Not Useful

Let's turn first to the setting in which inventories aren't needed.

In this problem, the firm forms an output plan that maximizes the expected value of

$$\sum_{t=0}^{\infty} \beta^t \{p_t Q_t - C(Q_t)\}$$

It turns out that the optimal plan for Q_t for this problem also solves a sequence of static problems $\max_{Q_t}\{p_t Q_t - c(Q_t)\}$.

When inventories aren't required or used, sales always equal production.

This simplifies the problem and the optimal no-inventory production maximizes the expected value of

$$\sum_{t=0}^{\infty} \beta^t \left\{p_t Q_t - C\left(Q_t\right)\right\}.$$

The optimum decision rule is

$$Q_t^{ni} = \frac{a_0 + \nu_t - c_1}{c_2 + a_1}.$$

9.4 Inventories Useful but are Hardwired to be Zero Always

Next, we turn to a distinct problem in which inventories are useful – meaning that there are costs of $d_2(I_t - S_t)^2$ associated with having sales not equal to inventories – but we arbitrarily impose on the firm the costly restriction that it never hold inventories.

Here the firm's maximization problem is

$$\max_{\{I_t, Q_t, S_t\}} \sum_{t=0}^{\infty} \beta^t \left\{p_t S_t - C\left(Q_t\right) - d\left(I_t, S_t\right)\right\}$$

subject to the restrictions that $I_t = 0$ for all t and that $I_{t+1} = I_t + Q_t - S_t$.

The restriction that $I_t = 0$ implies that $Q_t = S_t$ and that the maximization problem reduces to

$$\max_{Q_t} \sum_{t=0}^{\infty} \beta^t \left\{p_t Q_t - C\left(Q_t\right) - d\left(0, Q_t\right)\right\}$$

Here the optimal production plan is

$$Q_t^h = \frac{a_0 + \nu_t - c_1}{c_2 + a_1 + d_2}.$$

We introduce this I_t **is hardwired to zero** specification in order to shed light on the role that inventories play by comparing outcomes with those under our two other versions of the problem.

The bottom right panel displays a production path for the original problem that we are interested in (the blue line) as well with an optimal production path for the model in which inventories are not useful (the green path) and also for the model in which, although inventories are useful, they are hardwired to zero and the firm pays cost $d(0, Q_t)$ for not setting sales $S_t = Q_t$ equal to zero (the orange line).

Notice that it is typically optimal for the firm to produce more when inventories aren't useful. Here there is no requirement to sell out of inventories and no costs from having sales deviate from inventories.

But “typical” does not mean “always”.

Thus, if we look closely, we notice that for small t, the green “production when inventories aren’t useful” line in the lower right panel is below optimal production in the original model.

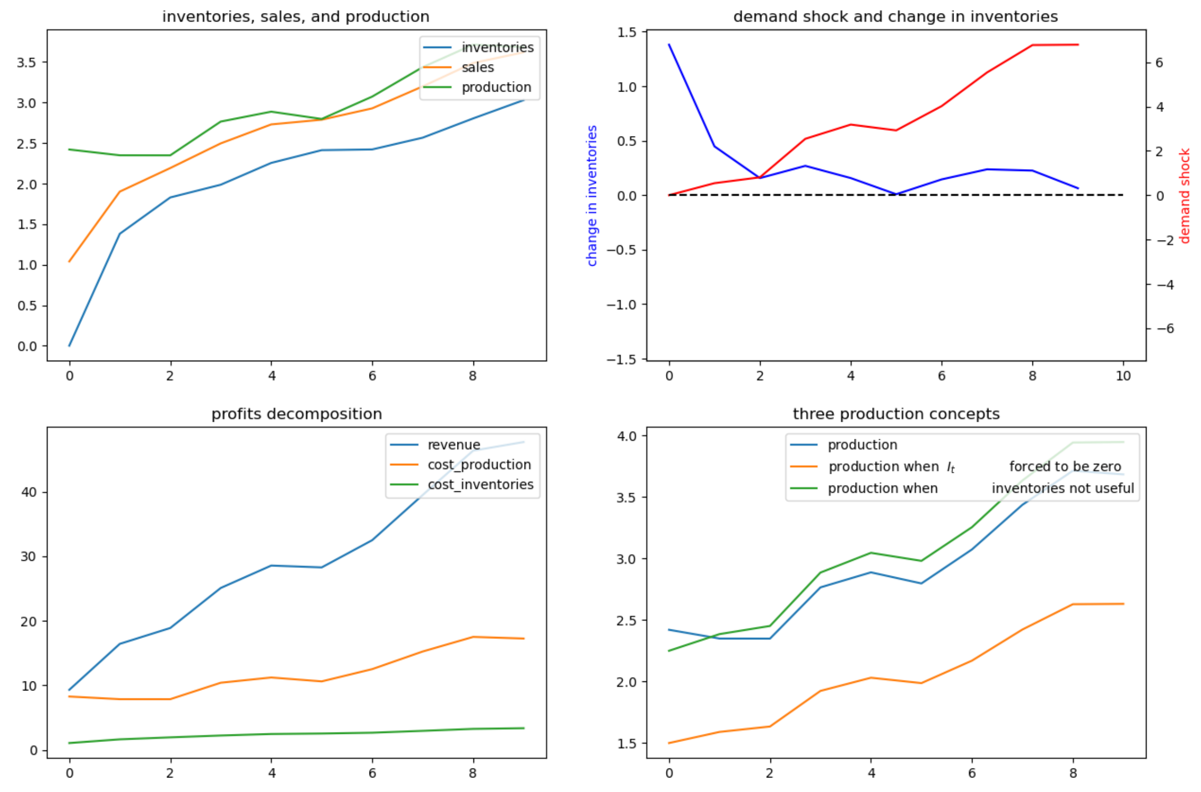
High optimal production in the original model early on occurs because the firm wants to accumulate inventories quickly in order to acquire high inventories for use in later periods.

But how the green line compares to the blue line early on depends on the evolution of the demand shock, as we will see in a deterministically seasonal demand shock example to be analyzed below.

In that example, the original firm optimally accumulates inventories slowly because the next positive demand shock is in the distant future.

To make the green-blue model production comparison easier to see, let’s confine the graphs to the first 10 periods:

```
ex1.simulate(x0, T=10)
```



9.5 Example 2

Next, we shut down randomness in demand and assume that the demand shock ν_t follows a deterministic path:

$$\nu_t = \alpha + \rho \nu_{t-1}$$

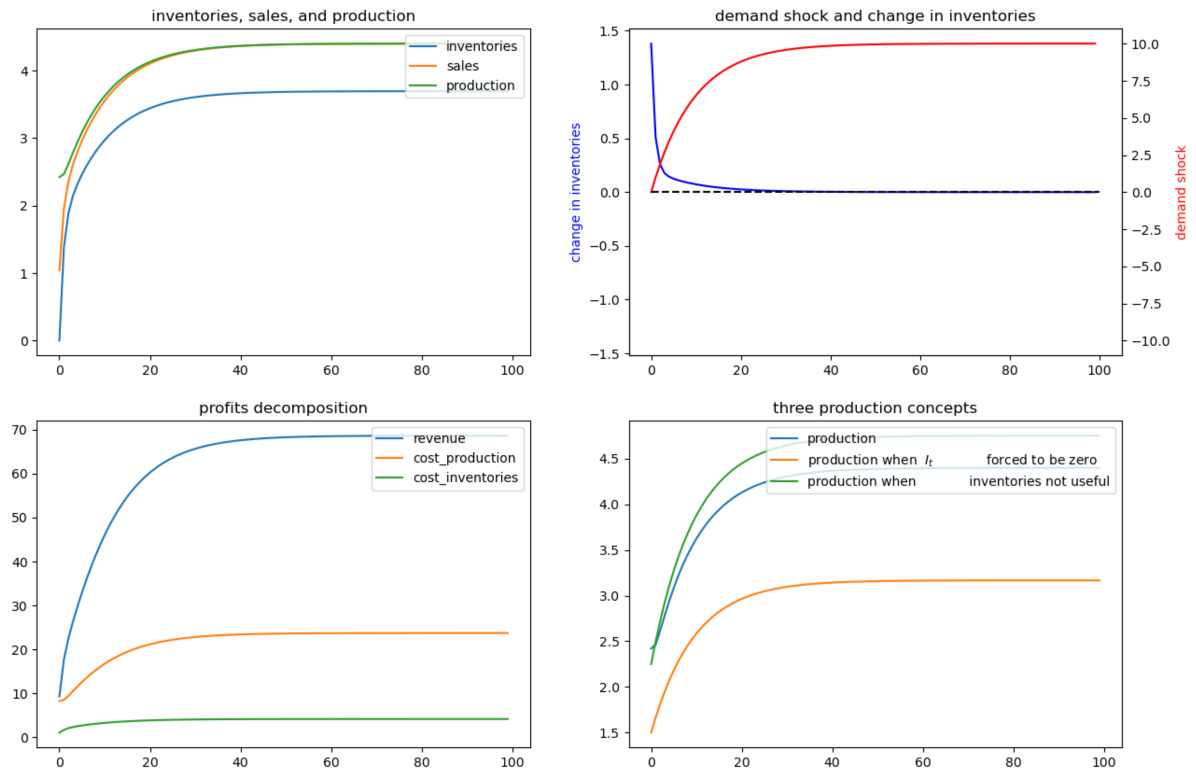
Again, we’ll compute and display outcomes in some figures

```
ex2 = SmoothingExample(C2=[[0], [0]])
```

(continues on next page)

(continued from previous page)

```
x0 = [0, 1, 0]
ex2.simulate(x0)
```



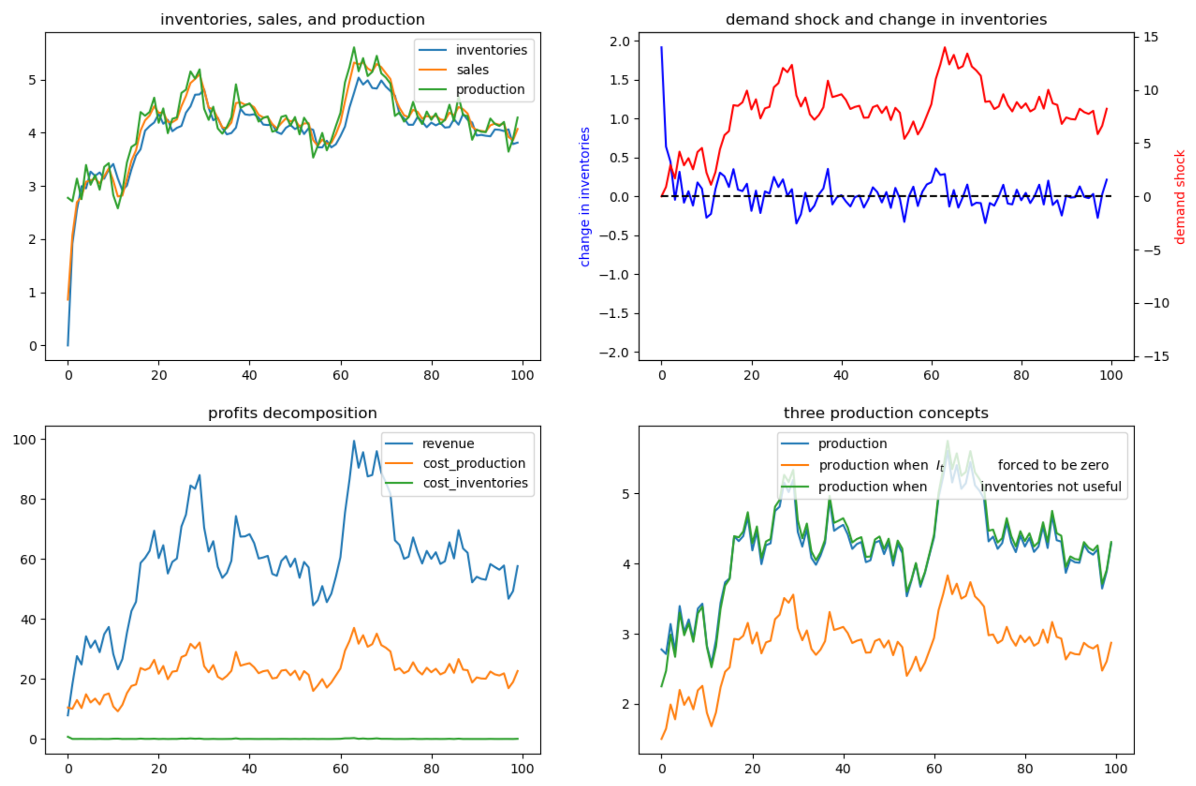
9.6 Example 3

Now we'll put randomness back into the demand shock process and also assume that there are zero costs of holding inventories.

In particular, we'll look at a situation in which $d_1 = 0$ but $d_2 > 0$.

Now it becomes optimal to set sales approximately equal to inventories and to use inventories to smooth production quite well, as the following figures confirm

```
ex3 = SmoothingExample(d1=0)

x0 = [0, 1, 0]
ex3.simulate(x0)
```



9.7 Example 4

To bring out some features of the optimal policy that are related to some technical issues in linear control theory, we'll now temporarily assume that it is costless to hold inventories.

When we completely shut down the cost of holding inventories by setting $d_1 = 0$ and $d_2 = 0$, something absurd happens (because the Bellman equation is opportunistic and very smart).

(Technically, we have set parameters that end up violating conditions needed to assure **stability** of the optimally controlled state.)

The firm finds it optimal to set $Q_t \equiv Q^* = \frac{-c_1}{2c_2}$, an output level that sets the costs of production to zero (when $c_1 > 0$, as it is with our default settings, then it is optimal to set production negative, whatever that means!).

Recall the law of motion for inventories

$$I_{t+1} = I_t + Q_t - S_t$$

So when $d_1 = d_2 = 0$ so that the firm finds it optimal to set $Q_t = \frac{-c_1}{2c_2}$ for all t, then

$$I_{t+1} - I_t = \frac{-c_1}{2c_2} - S_t < 0$$

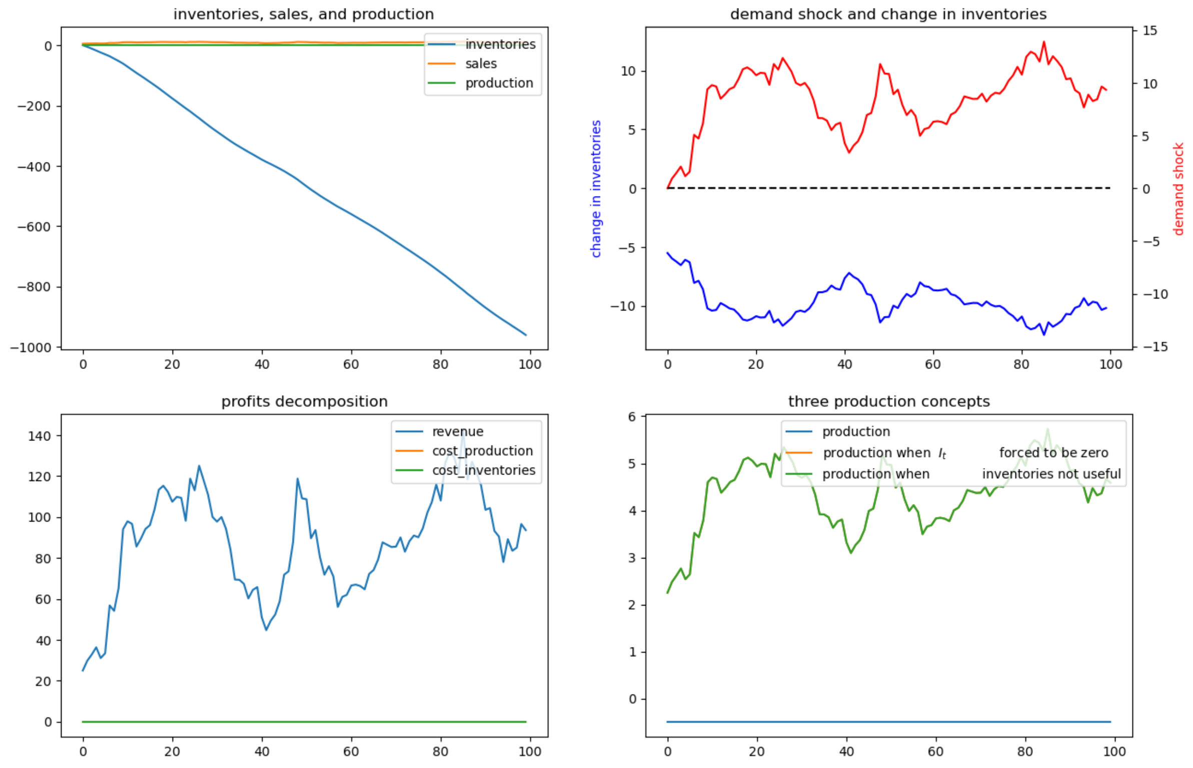
for almost all values of S_t under our default parameters that keep demand positive almost all of the time.

The dynamic program instructs the firm to set production costs to zero and to **run a Ponzi scheme** by running inventories down forever.

(We can interpret this as the firm somehow **going short in** or **borrowing** inventories)

The following figures confirm that inventories head south without limit

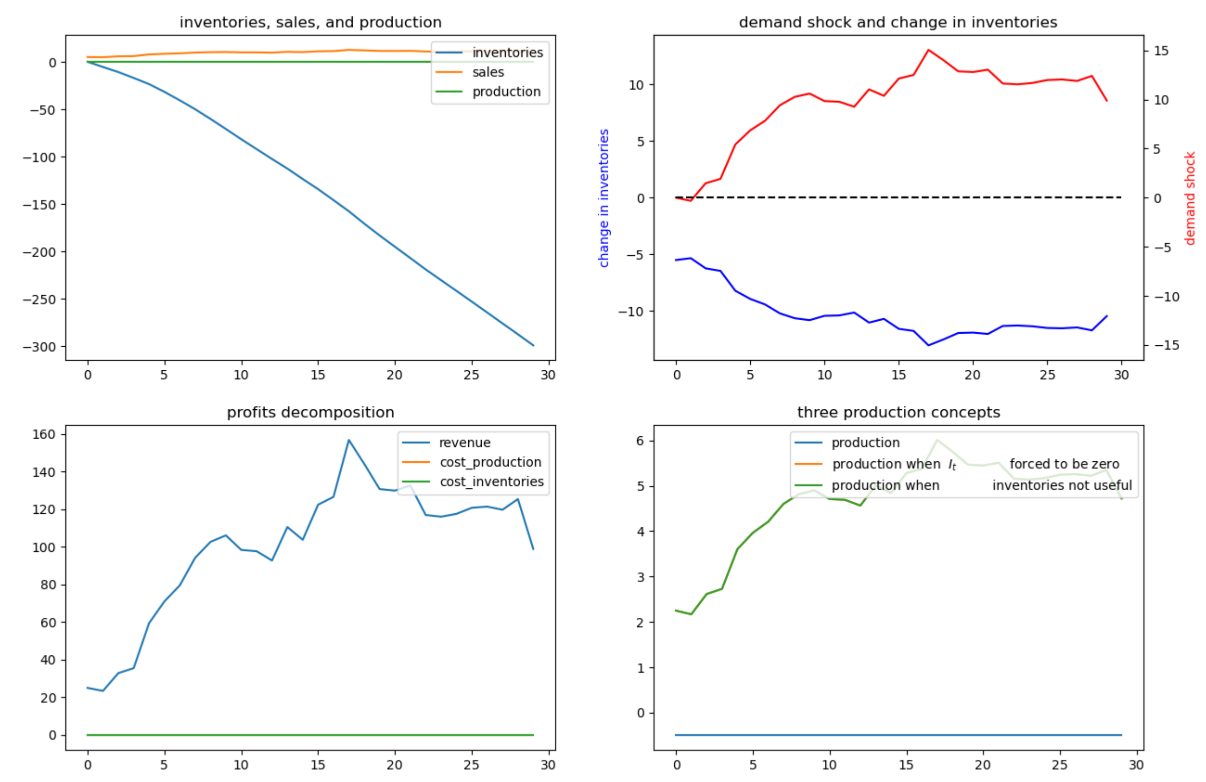
```
ex4 = SmoothingExample(d1=0, d2=0)

x0 = [0, 1, 0]
ex4.simulate(x0)
```



Let's shorten the time span displayed in order to highlight what is going on.

We'll set the horizon $T = 30$ with the following code

```
# shorter period
ex4.simulate(x0, T=30)
```



9.8 Example 5

Now we'll assume that the demand shock that follows a linear time trend

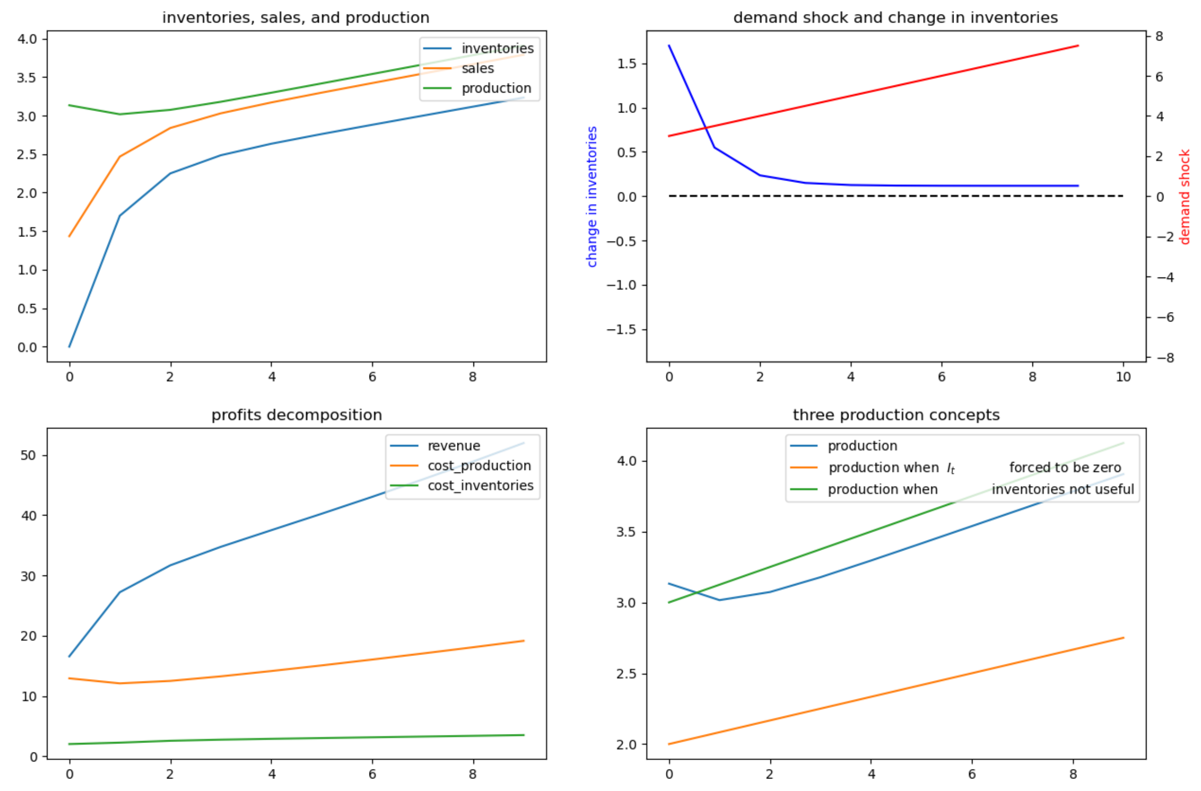
$$v_t = b + at, a > 0, b > 0$$

To represent this, we set $C_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and

$$A_{22} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, G = \begin{bmatrix} b & a \end{bmatrix}$$

```
# Set parameters
a = 0.5
b = 3.
```

```
ex5 = SmoothingExample(A22=[[1, 0], [1, 1]], C2=[[0], [0]], G=[b, a])

x0 = [0, 1, 0] # set the initial inventory as 0
ex5.simulate(x0, T=10)
```



9.9 Example 6

Now we'll assume a deterministically seasonal demand shock.

To represent this we'll set

$$A_{22} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, C_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, G' = \begin{bmatrix} b \\ a \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where $a > 0, b > 0$ and

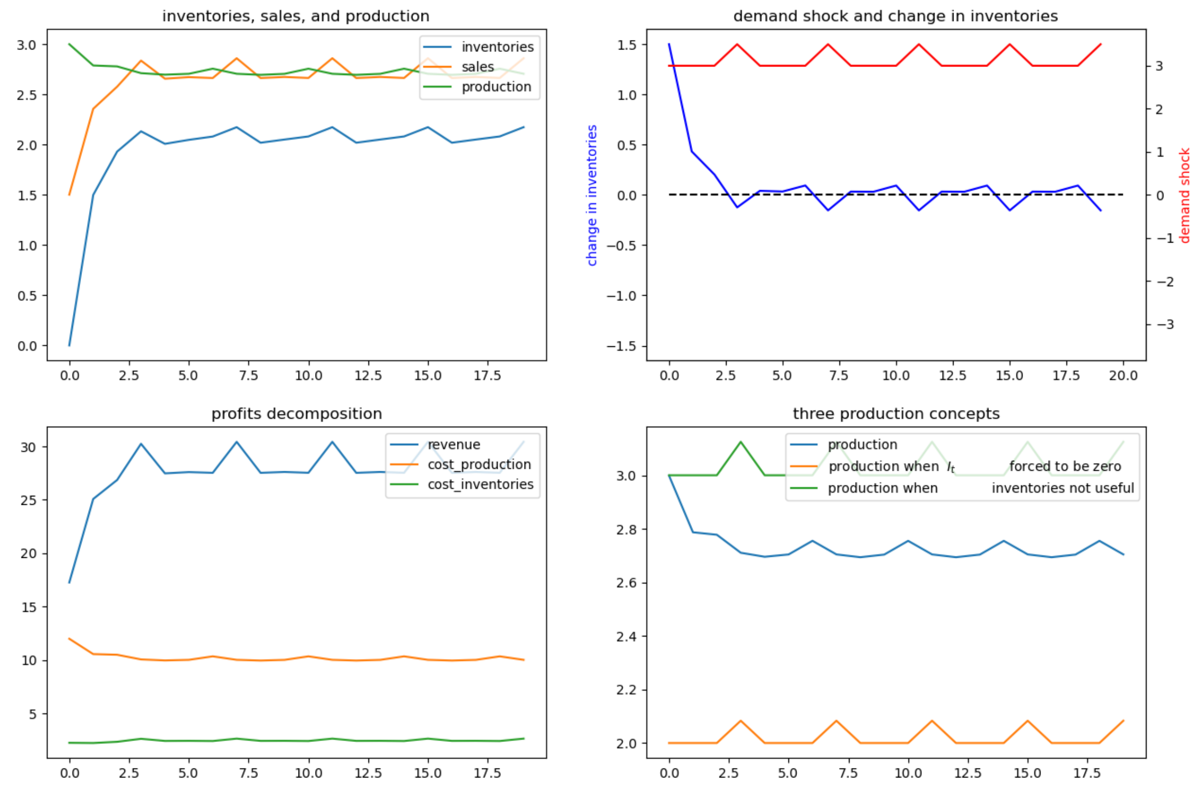
$$x_0 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

```
ex6 = SmoothingExample(A22=[[1, 0, 0, 0, 0],
                            [0, 0, 0, 0, 1],
                            [0, 1, 0, 0, 0],
                            [0, 0, 1, 0, 0],
                            [0, 0, 0, 1, 0]],
                       C2=[[0], [0], [0], [0], [0]],
                       G=[b, a, 0, 0, 0])
```

(continues on next page)

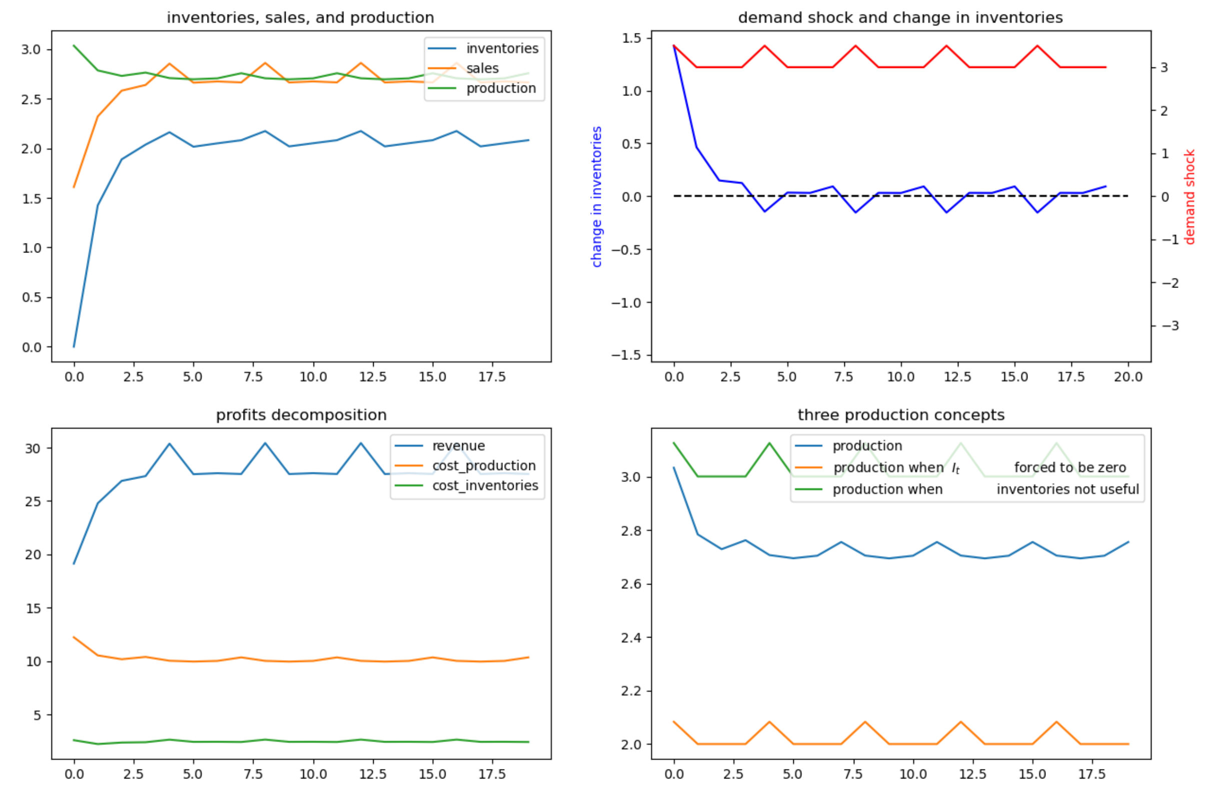
(continued from previous page)

```
x00 = [0, 1, 0, 1, 0, 0] # Set the initial inventory as 0
ex6.simulate(x00, T=20)
```

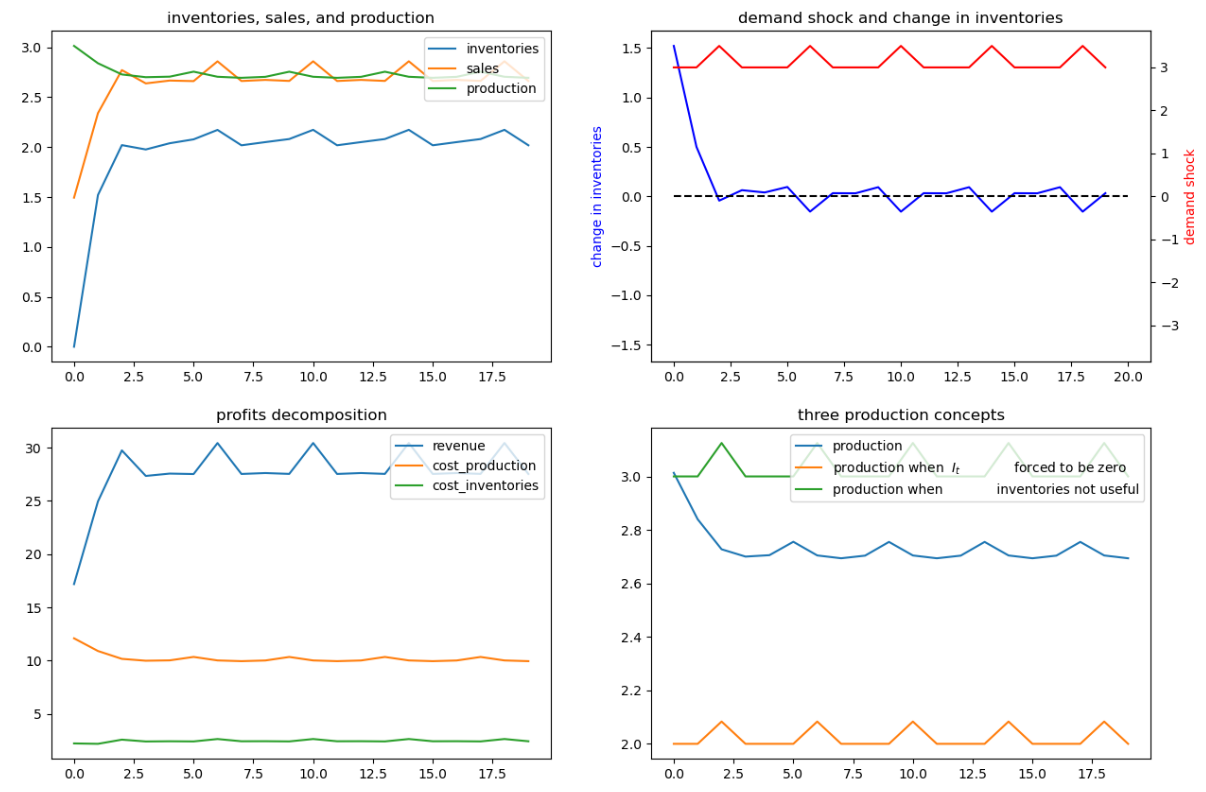


Now we'll generate some more examples that differ simply from the initial **season** of the year in which we begin the demand shock

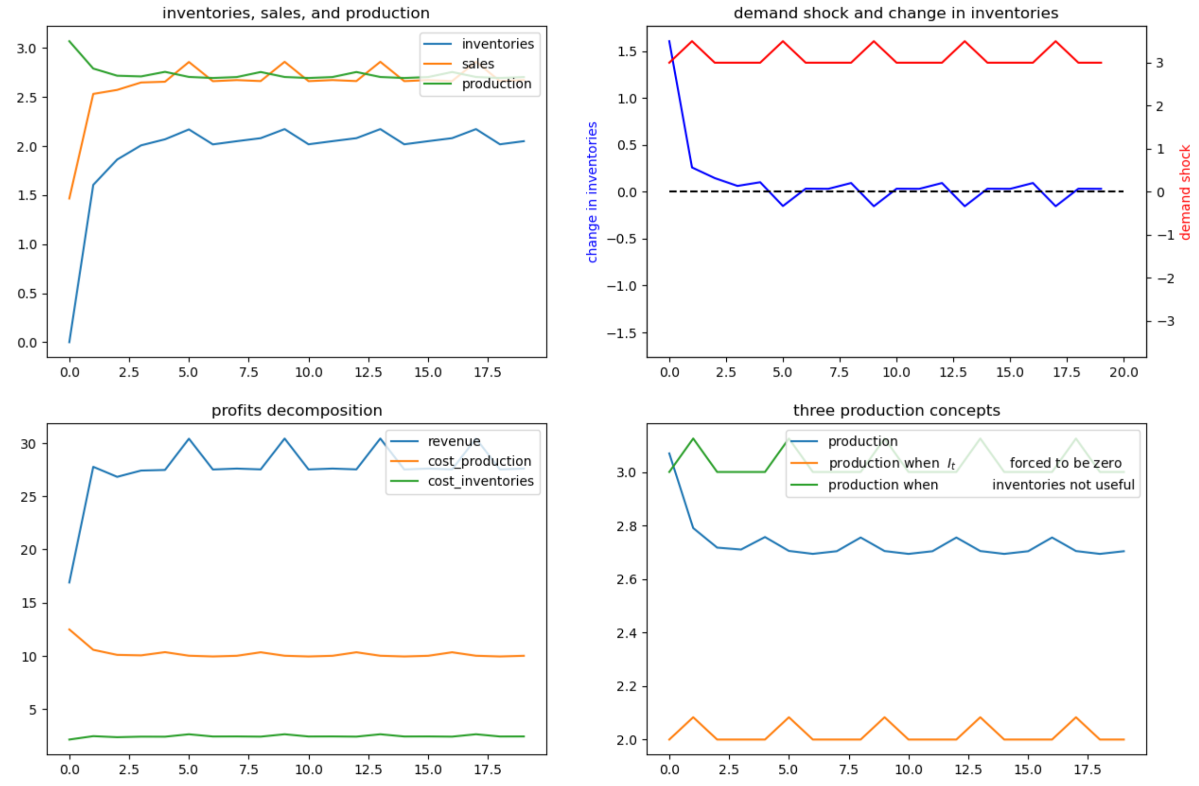
```
x01 = [0, 1, 1, 0, 0, 0]
ex6.simulate(x01, T=20)
```



```
x02 = [0, 1, 0, 0, 1, 0]
ex6.simulate(x02, T=20)
```



```
x03 = [0, 1, 0, 0, 0, 1]
ex6.simulate(x03, T=20)
```



9.10 Exercises

Please try to analyze some inventory sales smoothing problems using the `SmoothingExample` class.

Exercise 9.10.1

Assume that the demand shock follows AR(2) process below:

$$\nu_t = \alpha + \rho_1 \nu_{t-1} + \rho_2 \nu_{t-2} + \epsilon_t.$$

where $\alpha = 1$, $\rho_1 = 1.2$, and $\rho_2 = -0.3$. You need to construct $A22$, C, and G matrices properly and then to input them as the keyword arguments of `SmoothingExample` class. Simulate paths starting from the initial condition $x_0 = [0, 1, 0, 0]'$.

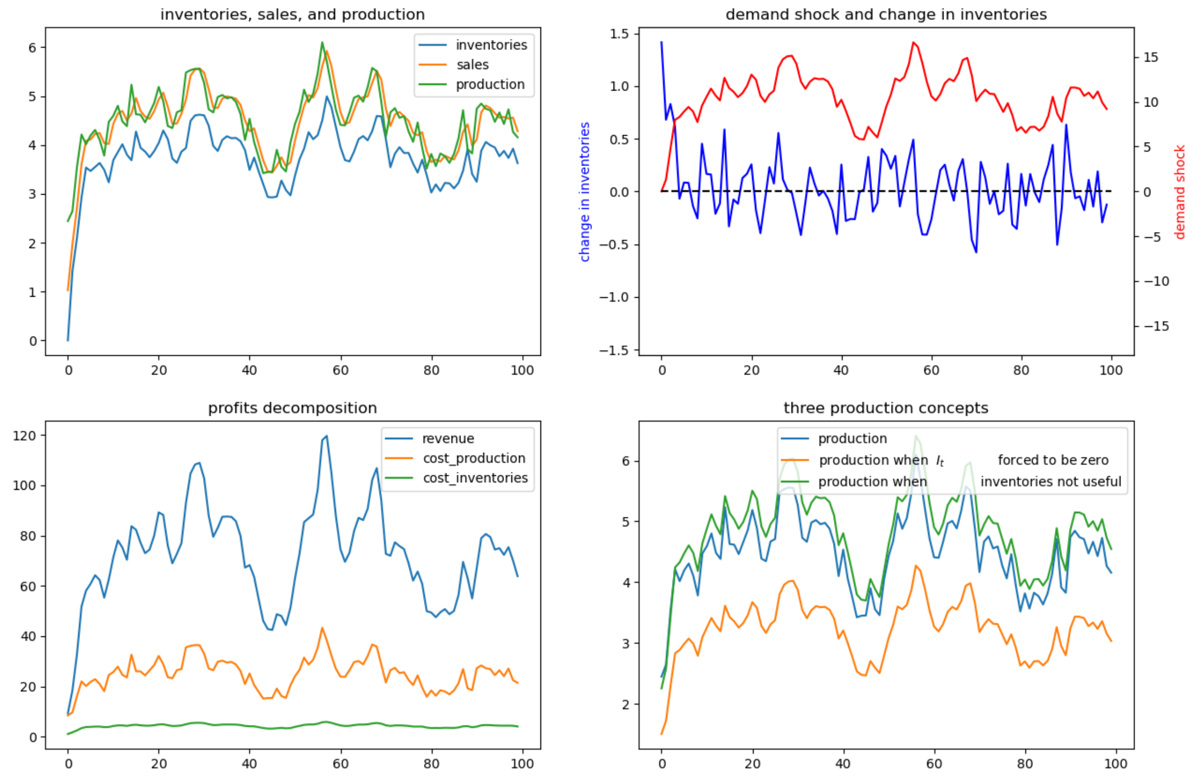
After this, try to construct a very similar `SmoothingExample` with the same demand shock process but exclude the randomness ϵ_t. Compute the stationary states $\bar{x}$ by simulating for a long period. Then try to add shocks with different magnitude to $\bar{\nu}_t$ and simulate paths. You should see how firms respond differently by staring at the production plans.

Solution to Exercise 9.10.1

```
# set parameters
α = 1
ρ1 = 1.2
ρ2 = -.3
```

```
# construct matrices
A22 =[[1,  0,  0],
          [1, ρ1, ρ2],
          [0,  1, 0]]
C2 = [[0], [1], [0]]
G = [0, 1, 0]
```

```
ex1 = SmoothingExample(A22=A22, C2=C2, G=G)

x0 = [0, 1, 0, 0] # initial condition
ex1.simulate(x0)
```



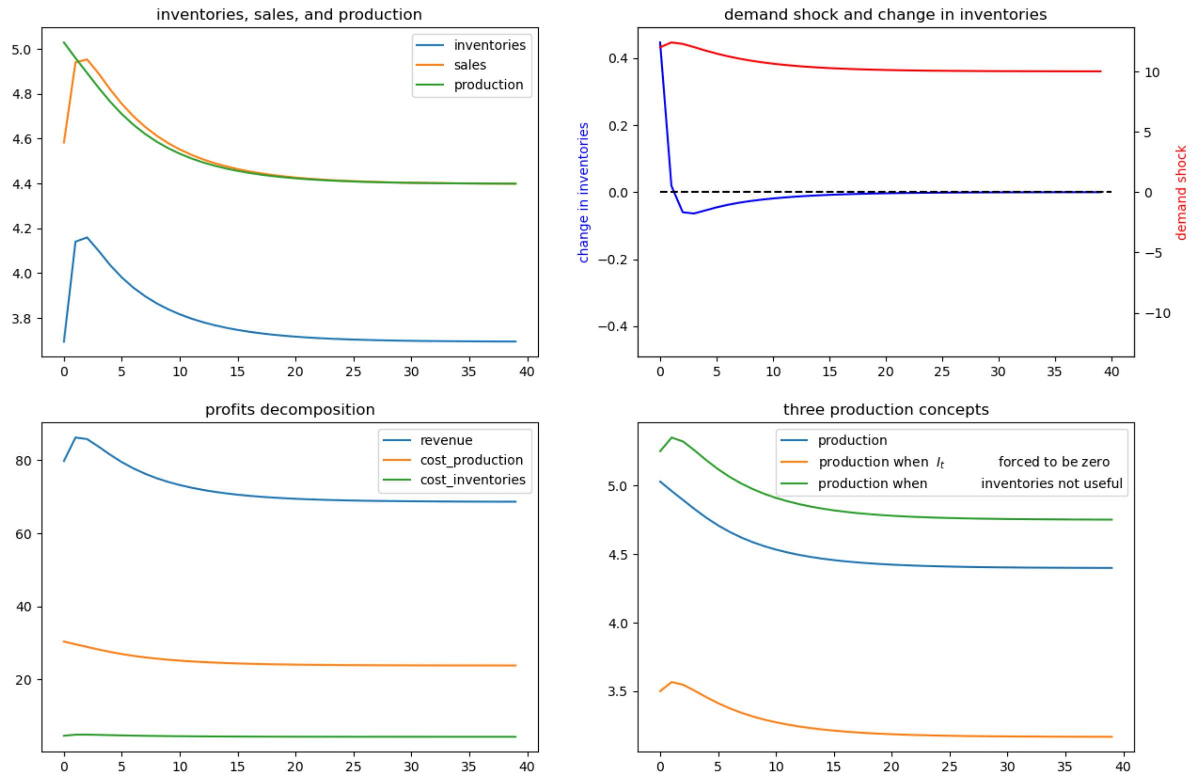
```
# now silence the noise
ex1_no_noise = SmoothingExample(A22=A22, C2=[[0], [0], [0]], G=G)

# initial condition
x0 = [0, 1, 0, 0]

# compute stationary states
x_bar = ex1_no_noise.LQ.compute_sequence(x0, ts_length=250)[0][:, -1]
x_bar
```

```
array([ 3.69387755,  1.        , 10.        , 10.        ])
```

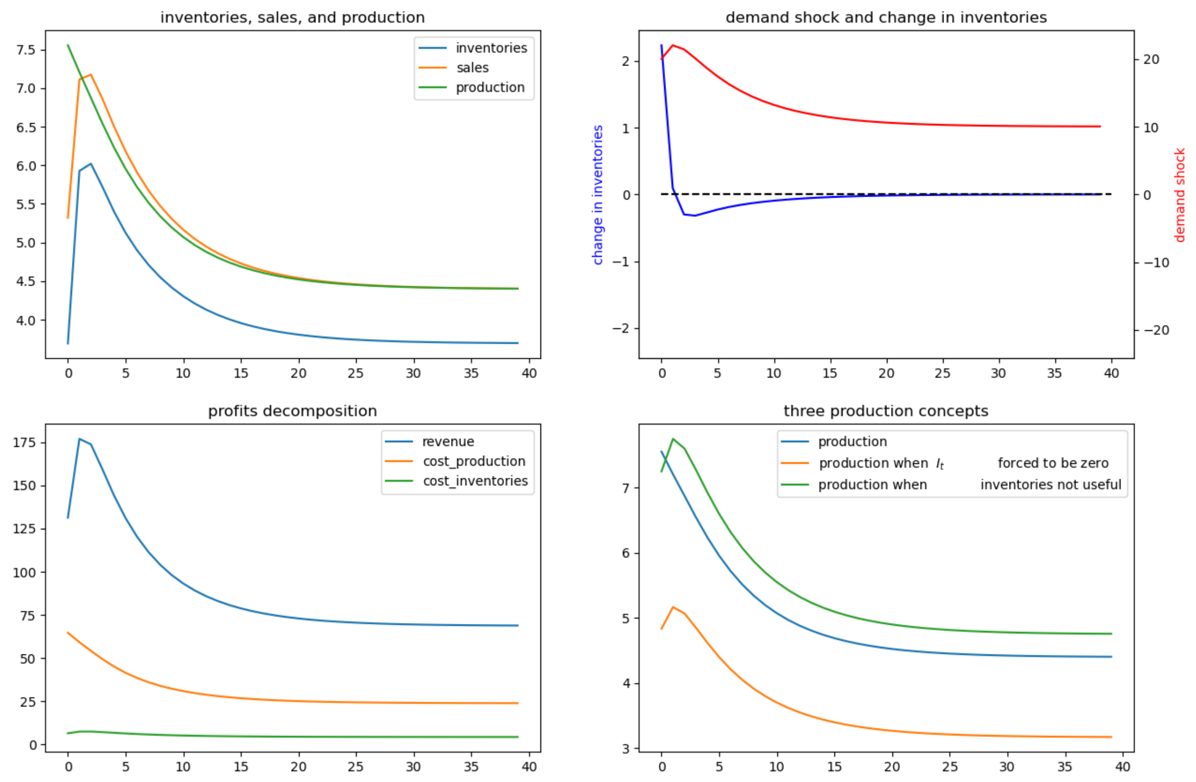
In the following, we add small and large shocks to $\bar{\nu}_t$ and compare how firm responds differently in quantity. As the shock is not very persistent under the parameterization we are using, we focus on a short period response.

```
T = 40
```

```
# small shock
x_bar1 = x_bar.copy()
x_bar1[2] += 2
ex1_no_noise.simulate(x_bar1, T=T)
```



```
# large shock
x_bar1 = x_bar.copy()
x_bar1[2] += 10
ex1_no_noise.simulate(x_bar1, T=T)
```



Exercise 9.10.2

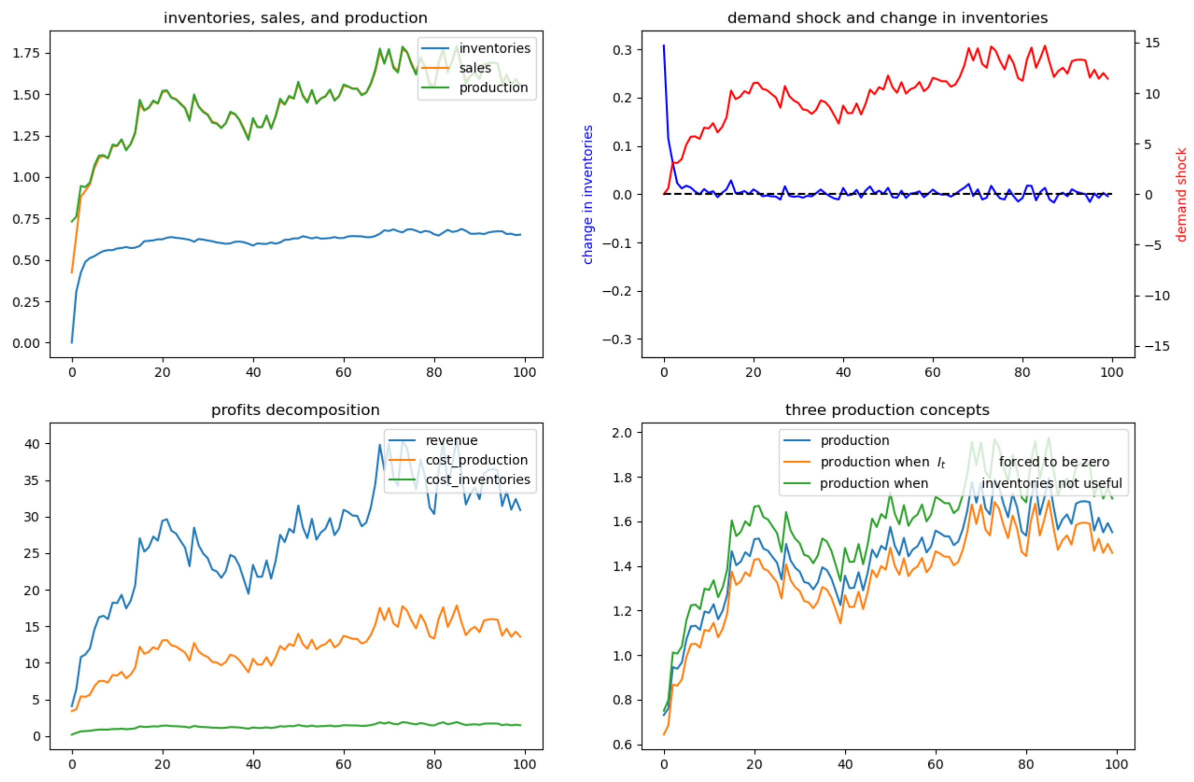
Change parameters of $C(Q_t)$ and $d(I_t, S_t)$.

1. Make production more costly, by setting $c_2 = 5$.
2. Increase the cost of having inventories deviate from sales, by setting $d_2 = 5$.

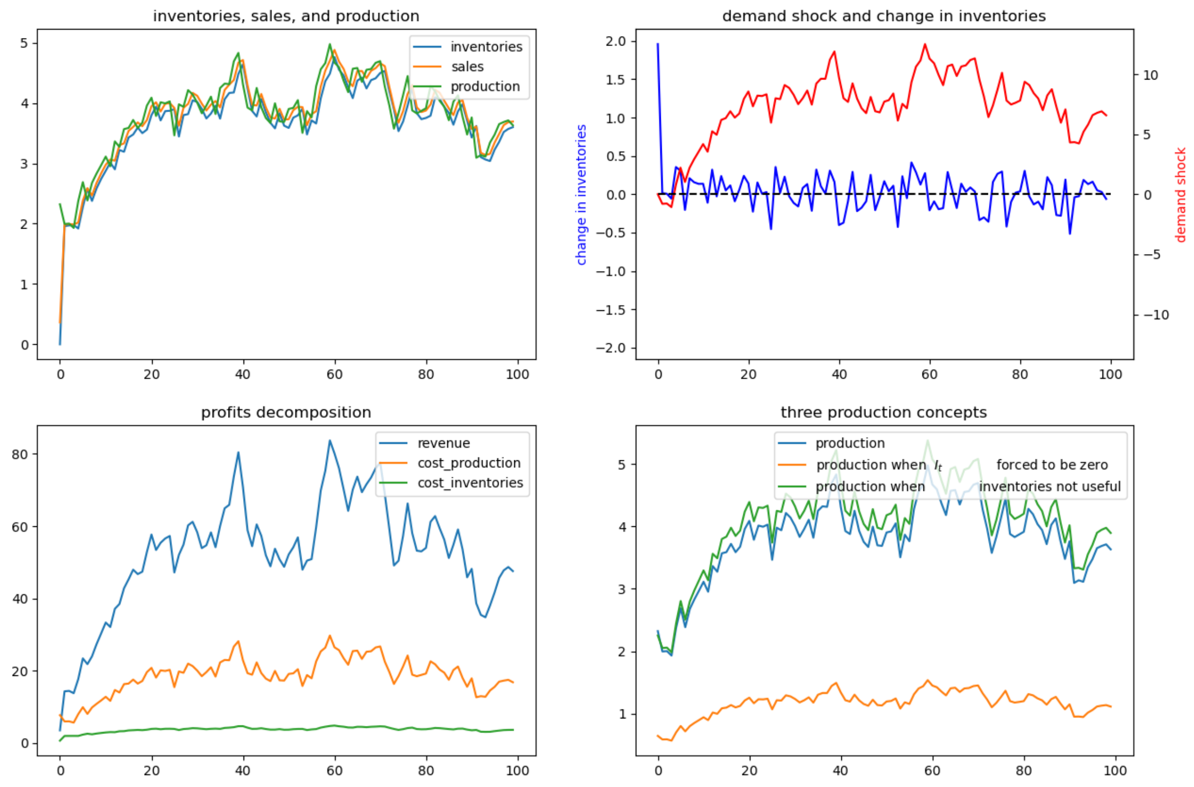
Solution to Exercise 9.10.2

```
x0 = [0, 1, 0]
```

```
SmoothingExample(c2=5).simulate(x0)
```



```
SmoothingExample(d2=5).simulate(x0)
```



CHAPTER

TEN

INFORMATION AND CONSUMPTION SMOOTHING

Contents

In addition to what's in Anaconda, this lecture employs the following libraries:

```
!pip install --upgrade quantecon
```

10.1 Overview

In the linear-quadratic permanent income of consumption smoothing model described in this quantecon lecture, a scalar parameter $\beta \in (0, 1)$ plays two roles:

- it is a **discount factor** that the consumer applies to future utilities from consumption
- it is the reciprocal of the gross **interest rate** on risk-free one-period loans

That β plays these two roles is essential in delivering the outcome that, **regardless** of the stochastic process that describes his non-financial income, the consumer chooses to make consumption follow a random walk (see [Hall, 1978]).

In this lecture, we assign a third role to β:

- it describes a **first-order moving average** process for the growth in non-financial income

10.1.1 Same non-financial incomes, different information

We study two consumers who have exactly the same nonfinancial income process and who both conform to the linear-quadratic permanent income of consumption smoothing model described here.

The two consumers have different information about their future nonfinancial incomes.

A better informed consumer each period receives **news** in the form of a shock that simultaneously affects both **today's** nonfinancial income and the present value of **future** nonfinancial incomes in a particular way.

A less informed consumer each period receives a shock that equals the part of today's nonfinancial income that could not be forecast from past values of nonfinancial income.

Even though they receive exactly the same nonfinancial incomes each period, our two consumers behave differently because they have different information about their future nonfinancial incomes.

The second consumer receives less information about future nonfinancial incomes in a sense that we shall make precise.

This difference in their information sets manifests itself in their responding differently to what they regard as time t **information shocks**.

Thus, although at each date they receive exactly the same histories of nonfinancial income, our two consumers receive different **shocks** or **news** about their **future** nonfinancial incomes.

We use the different behaviors of our consumers as a way to learn about

- operating characteristics of a linear-quadratic permanent income model
- how the Kalman filter introduced in this lecture and/or another representation of the theory of optimal forecasting introduced in this lecture embody lessons that can be applied to the **news** and **noise** literature
- ways of representing and computing optimal decision rules in the linear-quadratic permanent income model
- a **Ricardian equivalence** outcome that describes effects on optimal consumption of a tax cut at time t accompanied by a foreseen permanent increases in taxes that is just sufficient to cover the interest payments used to service the risk-free government bonds that are issued to finance the tax cut
- a simple application of alternative ways to factor a covariance generating function along lines described in this lecture

This lecture can be regarded as an introduction to **invertibility** issues that take center stage in the analysis of **fiscal foresight** by Eric Leeper, Todd Walker, and Susan Yang [Leeper *et al.*, 2013], as well as in chapter 4 of [Sargent *et al.*, 1991].

10.2 Two Representations of One Nonfinancial Income Process

We study consequences of endowing a consumer with one of two alternative representations for the change in the consumer's nonfinancial income $y_{t+1} - y_t$.

For both types of consumer, a parameter $\beta \in (0, 1)$ plays three roles.

It appears

- as a **discount factor** applied to future expected one-period utilities,
- as the **reciprocal of a gross interest rate** on one-period loans, and
- as a parameter in a first-order moving average that equals the increment in a consumer's non-financial income

The first representation, which we shall sometimes refer to as the **more informative representation**, is

$$y_{t+1} - y_t = \epsilon_{t+1} - \beta^{-1}\epsilon_t \tag{10.1}$$

where $\{\epsilon_t\}$ is an i.i.d. normally distributed scalar process with means of zero and contemporaneous variances σ_ϵ^2.

This representation of the process is used by a consumer who at time t knows both y_t and the shock ϵ_t and can use both of them to forecast future y_{t+j}'s.

As we'll see below, representation (10.1) has the peculiar property that a positive shock ϵ_{t+1} leaves the discounted present value of the consumer's financial income at time $t+1$ unaltered.

The second representation of the **same** $\{y_t\}$ process is

$$y_{t+1} - y_t = a_{t+1} - \beta a_t \tag{10.2}$$

where $\{a_t\}$ is another i.i.d. normally distributed scalar process, with means of zero and now variances $\sigma_a^2 > \sigma_\epsilon^2$.

The i.i.d. shock variances are related by

$$\sigma_a^2 = \beta^{-2}\sigma_\epsilon^2 > \sigma_\epsilon^2$$

so that the variance of the innovation exceeds the variance of the original shock by a multiplicative factor β^{-2}.

Representation (10.2) is the **innovations representation** of equation (10.1) associated with Kalman filtering theory.

To see how this works, note that equating representations (10.1) and (10.2) for $y_{t+1} - y_t$ implies $\epsilon_{t+1} - \beta^{-1}\epsilon_t = a_{t+1} - \beta a_t$, which in turn implies

$$a_{t+1} = \beta a_t + \epsilon_{t+1} - \beta^{-1}\epsilon_t.$$

Solving this difference equation backwards for a_{t+1} gives, after a few lines of algebra,

$$a_{t+1} = \epsilon_{t+1} + (\beta - \beta^{-1})\sum_{j=0}^{\infty}\beta^j \epsilon_{t-j} \tag{10.3}$$

which we can also write as

$$a_{t+1} = \sum_{j=0}^{\infty} h_j \epsilon_{t+1-j} \equiv h(L)\epsilon_{t+1}$$

where L is the one-period lag operator, $h(L) = \sum_{j=0}^{\infty} h_j L^j$, I is the identity operator, and

$$h(L) = \frac{I - \beta^{-1}L}{I - \beta L}$$

Let $g_j \equiv E z_t z_{t-j}$ be the jth autocovariance of the $\{y_t - y_{t-1}\}$ process.

Using calculations in the quantecon lecture, where $z \in C$ is a complex variable, the **covariance generating function** $g(z) = \sum_{j=-\infty}^{\infty} g_j z^j$ of the $\{y_t - y_{t-1}\}$ process equals

$$g(z) = \sigma_\epsilon^2 h(z)h(z^{-1}) = \beta^{-2}\sigma_\epsilon^2 > \sigma_\epsilon^2,$$

which confirms that $\{a_t\}$ is a **serially uncorrelated** process with variance

$$\sigma_a^2 = \beta^{-1}\sigma_\epsilon^2.$$

To verify these claims, just notice that $g(z) = \beta^{-2}\sigma_\epsilon^2$ implies that

- $g_0 = \beta^{-2}\sigma_\epsilon^2$, and

- $g_j = 0$ for $j \neq 0$.

Alternatively, if you are uncomfortable with covariance generating functions, note that we can directly calculate σ_a^2 from formula (10.3) according to

$$\sigma_a^2 = \sigma_\epsilon^2 + [1 + (\beta - \beta^{-1})^2 \sum_{j=0}^{\infty} \beta^{2j}] = \beta^{-1}\sigma_\epsilon^2.$$

10.3 Application of Kalman filter

We can also use the the **Kalman filter** to obtain representation (10.2) from representation (10.1).

Thus, from equations associated with the **Kalman filter**, it can be verified that the steady-state Kalman gain $K = \beta^2$ and the steady state conditional covariance

$$\Sigma = E[(\epsilon_t - \hat{\epsilon}_t)^2 | y_{t-1}, y_{t-2}, ...] = (1 - \beta^2)\sigma_\epsilon^2$$

In a little more detail, let $z_t = y_t - y_{t-1}$ and form the state-space representation

$$\begin{aligned}\epsilon_{t+1} &= 0\epsilon_t + \epsilon_{t+1} \\ z_{t+1} &= -\beta^{-1}\epsilon_t + \epsilon_{t+1}\end{aligned}$$

and assume that $\sigma_\epsilon = 1$ for convenience

Let's compute the steady-state Kalman filter for this system.

Let K be the steady-state gain and a_{t+1} the one-step ahead innovation.

The steady-state innovations representation is

$$\begin{aligned}\hat{\epsilon}_{t+1} &= 0\hat{\epsilon}_t + Ka_{t+1} \\ z_{t+1} &= -\beta a_t + a_{t+1}\end{aligned}$$

By applying formulas for the steady-state Kalman filter, by hand it is possible to verify that $K = \beta^2, \sigma_a^2 = \beta^{-2}\sigma_\epsilon^2 = \beta^{-2}$, and $\Sigma = (1 - \beta^2)\sigma_\epsilon^2$.

Alternatively, we can obtain these formulas via the classical filtering theory described in this lecture.

10.4 News Shocks and Less Informative Shocks

Representation (10.1) is cast in terms of a **news shock** ϵ_{t+1} that represents a shock to nonfinancial income coming from taxes, transfers, and other random sources of income changes known to a well-informed person who perhaps has all sorts of information about the income process.

Representation (10.2) for the **same** income process is driven by shocks a_t that contain less information than the news shock ϵ_t.

Representation (10.2) is called the **innovations** representation for the $\{y_t - y_{t-1}\}$ process.

It is cast in terms of what time series statisticians call the **innovation** or **fundamental** shock that emerges from applying the theory of optimally predicting nonfinancial income based solely on the information in **past** levels of growth in nonfinancial income.

Fundamental for the y_t process means that the shock a_t can be expressed as a square-summable linear combination of $y_t, y_{t-1}, ...$

The shock ϵ_t is **not fundamental** because it has more information about the future of the $\{y_t - y_{t-1}\}$ process than is contained in a_t.

Representation (10.3) reveals the important fact that the **original shock** ϵ_t contains more information about future y's than is contained in the semi-infinite history $y^t = [y_t, y_{t-1}, ...]$.

Staring at representation (10.3) for a_{t+1} shows that it consists both of **new news** ϵ_{t+1} as well as a long moving average $(\beta - \beta^{-1})\sum_{j=0}^{\infty}\beta^j\epsilon_{t-j}$ of **old news**.

The **more information** representation (10.1) asserts that a shock ϵ_t results in an impulse response to nonfinancial income of ϵ_t times the sequence

$$1, 1-\beta^{-1}, 1-\beta^{-1}, ...$$

so that a shock that **increases** nonfinancial income y_t by ϵ_t at time t is followed by a change in future y of ϵ_t times $1-\beta^{-1} < 0$ in **all** subsequent periods.

Because $1-\beta^{-1} < 0$, this means that a positive shock of ϵ_t today raises income at time t by ϵ_t and then permanently **decreases all** future incomes by $(\beta^{-1}-1)\epsilon_t$.

This pattern precisely describes the following mental experiment:

- The consumer receives a government transfer of ϵ_t at time t.
- The government finances the transfer by issuing a one-period bond on which it pays a gross one-period risk-free interest rate equal to β^{-1}.
- In each future period, the government **rolls over** the one-period bond and so continues to borrow ϵ_t forever.
- The government imposes a lump-sum tax on the consumer in order to pay just the current interest on the original bond and its rolled over successors.
- Thus, in periods $t+1, t+2, ...$, the government levies a lump-sum tax on the consumer of $\beta^{-1}-1$ that is just enough to pay the interest on the bond.

The **present value** of the impulse response or moving average coefficients equals $d_\epsilon(L) = \frac{0}{1-\beta} = 0$, a fact that we'll see again below.

Representation (10.2), i.e., the innovations representation, asserts that a shock a_t results in an impulse response to nonfinancial income of a_t times

$$1, 1-\beta, 1-\beta, ...$$

so that a shock that increases income y_t by a_t at time t can be expected to be followed by an **increase** in y_{t+j} of a_t times $1-\beta > 0$ in all future periods $j = 1, 2, ...$

The present value of the impulse response or moving average coefficients for representation (10.2) is $d_a(\beta) = \frac{1-\beta^2}{1-\beta} = (1+\beta)$, another fact that will be important below.

10.5 Representation of ϵ_t Shock in Terms of Future y_t

Notice that reprentation (10.1), namely, $y_{t+1} - y_t = -\beta^{-1}\epsilon_t + \epsilon_{t+1}$ implies the linear difference equation

$$\epsilon_t = \beta\epsilon_{t+1} - \beta(y_{t+1} - y_t).$$

Solving forward we obtain

$$\epsilon_t = \beta(y_t - (1-\beta)\sum_{j=0}^{\infty}\beta^j y_{t+j+1})$$

This equation shows that ϵ_t equals β times the one-step-backwards error in optimally **backcasting** y_t based on the semi-infinite **future** $y^t_+ \equiv [y_{t+1}, y_{t+2}, ...]$ via the **optimal backcasting formula**

$$E[y_t | y^t_+] = (1-\beta) \sum_{j=0}^{\infty} \beta^j y_{t+j+1}$$

Thus, ϵ_t **exactly** reveals the gap between y_t and $E[y_t | y^t_+]$.

10.6 Representation in Terms of a_t Shocks

Next notice that representation (10.2), namely, $y_{t+1} - y_t = -\beta a_t + a_{t+1}$ implies the linear difference equation

$$a_{t+1} = \beta a_t + (y_{t+1} - y_t)$$

Solving this equation backward establishes that the one-step-prediction error a_{t+1} is

$$a_{t+1} = y_{t+1} - (1-\beta) \sum_{j=0}^{\infty} \beta^j y_{t-j}.$$

Here the information set is $y^t = [y_t, y_{t-1}, ...]$ and a one step-ahead optimal prediction is

$$E[y_{t+1} | y^t] = (1-\beta) \sum_{j=0}^{\infty} \beta^j y_{t-j}$$

10.7 Permanent Income Consumption-Smoothing Model

When we computed optimal consumption-saving policies for our two representations (10.1) and (10.2) by using formulas obtained with the difference equation approach described in quantecon lecture, we obtained:

for a consumer having the information assumed in the news representation (10.1):

$$\begin{aligned} c_{t+1} - c_t &= 0 \\ b_{t+1} - b_t &= -\beta^{-1} \epsilon_t \end{aligned}$$

for a consumer having the more limited information associated with the innovations representation (10.2):

$$\begin{aligned} c_{t+1} - c_t &= (1-\beta^2) a_{t+1} \\ b_{t+1} - b_t &= -\beta a_t \end{aligned}$$

These formulas agree with outcomes from Python programs below that deploy state-space representations and dynamic programming.

Evidently, although they receive exactly the same histories of nonfinancial incomethe two consumers behave differently.

The better informed consumer who has the information sets associated with representation (10.1) responds to each shock ϵ_{t+1} by leaving his consumption unaltered and **saving** all of ϵ_{t+1} in anticipation of the permanently increased taxes that he will bear in order to service the permanent interest payments on the risk-free bonds that the government has presumably issued to pay for the one-time addition ϵ_{t+1} to his time $t+1$ nonfinancial income.

The less well informed consumer who has information sets associated with representation (10.2) responds to a shock a_{t+1} by increasing his consumption by what he perceives to be the **permanent** part of the increase in consumption and by increasing his **saving** by what he perceives to be the temporary part.

The behavior of the better informed consumer sharply illustrates the behavior predicted in a classic Ricardian equivalence experiment.

10.8 State Space Representations

We now cast our representations (10.1) and (10.2), respectively, in terms of the following two state space systems:

$$\begin{aligned} \begin{bmatrix} y_{t+1} \\ \epsilon_{t+1} \end{bmatrix} &= \begin{bmatrix} 1 & -\beta^{-1} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ \epsilon_t \end{bmatrix} + \begin{bmatrix} \sigma_\epsilon \\ \sigma_\epsilon \end{bmatrix} v_{t+1} \\ y_t &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ \epsilon_t \end{bmatrix} \end{aligned} \tag{10.4}$$

and

$$\begin{aligned} \begin{bmatrix} y_{t+1} \\ a_{t+1} \end{bmatrix} &= \begin{bmatrix} 1 & -\beta \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ a_t \end{bmatrix} + \begin{bmatrix} \sigma_a \\ \sigma_a \end{bmatrix} u_{t+1} \\ y_t &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ a_t \end{bmatrix} \end{aligned} \tag{10.5}$$

where $\{v_t\}$ and $\{u_t\}$ are both i.i.d. sequences of univariate standardized normal random variables.

These two alternative income processes are ready to be used in the framework presented in the section "Comparison with the Difference Equation Approach" in thid quantecon lecture.

All the code that we shall use below is presented in that lecture.

10.9 Computations

We shall use Python to form two state-space representations (10.4) and (10.5).

We set the following parameter values $\sigma_\epsilon = 1, \sigma_a = \beta^{-1}\sigma_\epsilon = \beta^{-1}$ where β is the **same** value as the discount factor in the household's problem in the LQ savings problem in the lecture.

For these two representations, we use the code in this lecture to

- compute optimal decision rules for c_t, b_t for the two types of consumers associated with our two representations of nonfinancial income
- use the value function objects P, d returned by the code to compute optimal values for the two representations when evaluated at the initial condition

$$x_0 = \begin{bmatrix} 10 \\ 0 \end{bmatrix}$$

for each representation.

- create instances of the LinearStateSpace class for the two representations of the $\{y_t\}$ process and use them to obtain impulse response functions of c_t and b_t to the respective shocks ϵ_t and a_t for the two representations.
- run simulations of $\{y_t, c_t, b_t\}$ of length T under both of the representations

We formulae the problem:

$$\min \sum_{t=0}^{\infty} \beta^t \left(c_t - \gamma\right)^2$$

subject to a sequence of constraints

$$c_t + b_t = \frac{1}{1+r} b_{t+1} + y_t, \quad t \geq 0$$

where y_t follows one of the representations defined above.

Define the control as $u_t \equiv c_t - \gamma$.

(For simplicity we can assume $\gamma = 0$ below because γ has no effect on the impulse response functions that interest us.)

The state transition equations under our two representations for the nonfinancial income process $\{y_t\}$ can be written as

$$\begin{bmatrix} y_{t+1} \\ \epsilon_{t+1} \\ b_{t+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & -\beta^{-1} & 0 \\ 0 & 0 & 0 \\ -(1+r) & 0 & 1+r \end{bmatrix}}_{\equiv A_1} \begin{bmatrix} y_t \\ \epsilon_t \\ b_t \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1+r \end{bmatrix}}_{\equiv B_1} \begin{bmatrix} c_t \end{bmatrix} + \underbrace{\begin{bmatrix} \sigma_\epsilon \\ \sigma_\epsilon \\ 0 \end{bmatrix}}_{\equiv C_1} \nu_{t+1},$$

and

$$\begin{bmatrix} y_{t+1} \\ a_{t+1} \\ b_{t+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & -\beta & 0 \\ 0 & 0 & 0 \\ -(1+r) & 0 & 1+r \end{bmatrix}}_{\equiv A_2} \begin{bmatrix} y_t \\ a_t \\ b_t \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1+r \end{bmatrix}}_{\equiv B_2} \begin{bmatrix} c_t \end{bmatrix} + \underbrace{\begin{bmatrix} \sigma_a \\ \sigma_a \\ 0 \end{bmatrix}}_{\equiv C_2} u_{t+1}.$$

As usual, we start by importing packages.

```
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt
```

```
# Set parameters
β, σϵ = 0.95, 1
σa = σϵ / β

R = 1 / β

# Payoff matrices are the same for two representations
RLQ = np.array([[0, 0, 0],
                [0, 0, 0],
                [0, 0, 1e-12]]) # put penalty on debt
QLQ = np.array([1.])
```

```
# More informative representation state transition matrices
ALQ1 = np.array([[1, -R, 0],
                 [0, 0, 0],
                 [-R, 0, R]])
BLQ1 = np.array([[0, 0, R]]).T
CLQ1 = np.array([[σϵ, σϵ, 0]]).T

# Construct and solve the LQ problem
LQ1 = qe.LQ(QLQ, RLQ, ALQ1, BLQ1, C=CLQ1, beta=β)
P1, F1, d1 = LQ1.stationary_values()
```

```
# The optimal decision rule for c
-F1
```

```
array([[ 1.  , -1.  , -0.05]])
```

Evidently, optimal consumption and debt decision rules for the consumer having news representation (10.1) are

$$\begin{aligned} c_t^* &= y_t - \epsilon_t - (1-\beta)\, b_t, \\ b_{t+1}^* &= \beta^{-1} c_t^* + \beta^{-1} b_t - \beta^{-1} y_t \\ &= \beta^{-1} y_t - \beta^{-1} \epsilon_t - (\beta^{-1} - 1)\, b_t + \beta^{-1} b_t - \beta^{-1} y_t \\ &= b_t - \beta^{-1} \epsilon_t. \end{aligned}$$

```
# Innovations representation
ALQ2 = np.array([[1, -β, 0],
                 [0,  0, 0],
                 [-R, 0, R]])
BLQ2 = np.array([[0, 0, R]]).T
CLQ2 = np.array([[σa, σa, 0]]).T

LQ2 = qe.LQ(QLQ, RLQ, ALQ2, BLQ2, C=CLQ2, beta=β)
P2, F2, d2 = LQ2.stationary_values()
```

```
-F2
```

```
array([[ 1.    , -0.9025, -0.05  ]])
```

For a consumer having access only to the information associated with the innovations representation (10.2), the optimal decision rules are

$$\begin{aligned} c_t^* &= y_t - \beta^2 a_t - (1-\beta)\, b_t, \\ b_{t+1}^* &= \beta^{-1} c_t^* + \beta^{-1} b_t - \beta^{-1} y_t \\ &= \beta^{-1} y_t - \beta a_t - (\beta^{-1} - 1)\, b_t + \beta^{-1} b_t - \beta^{-1} y_t \\ &= b_t - \beta a_t. \end{aligned}$$

Now we construct two Linear State Space models that emerge from using optimal policies of the form $u_t = -F x_t$.

Take the more informative original representation (10.1) as an example:

$$\begin{bmatrix} y_{t+1} \\ \epsilon_{t+1} \\ b_{t+1} \end{bmatrix} = (A_1 - B_1 F_1) \begin{bmatrix} y_t \\ \epsilon_t \\ b_t \end{bmatrix} + C_1 \nu_{t+1}$$

$$\begin{bmatrix} c_t \\ b_t \end{bmatrix} = \begin{bmatrix} -F_1 \\ S_b \end{bmatrix} \begin{bmatrix} y_t \\ \epsilon_t \\ b_t \end{bmatrix}$$

To have the Linear State Space model be of an innovations representation form (10.2), we can simply replace the corresponding matrices.

```
# Construct two Linear State Space models
Sb = np.array([0, 0, 1])

ABF1 = ALQ1 - BLQ1 @ F1
G1 = np.vstack([-F1, Sb])
LSS1 = qe.LinearStateSpace(ABF1, CLQ1, G1)

ABF2 = ALQ2 - BLQ2 @ F2
G2 = np.vstack([-F2, Sb])
LSS2 = qe.LinearStateSpace(ABF2, CLQ2, G2)
```

The following code computes impulse response functions of c_t and b_t.

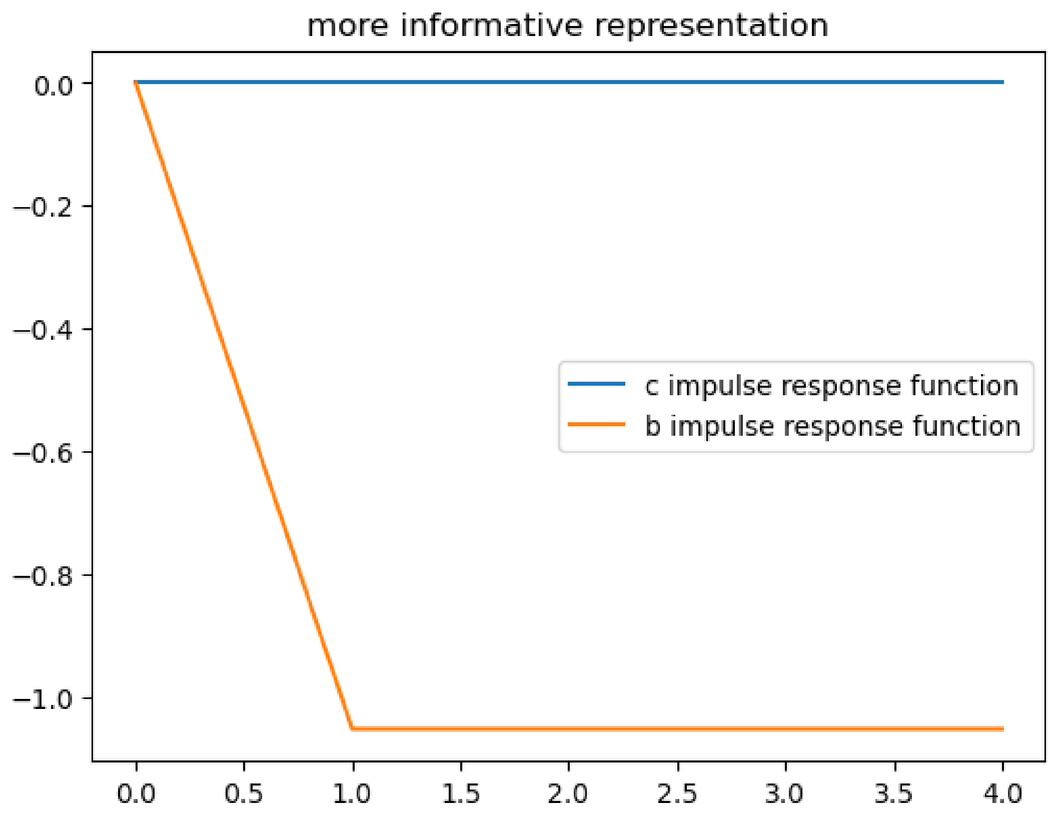
```
J = 5 # Number of coefficients that we want

x_res1, y_res1 = LSS1.impulse_response(j=J)
b_res1 = np.array([x_res1[i][2, 0] for i in range(J)])
c_res1 = np.array([y_res1[i][0, 0] for i in range(J)])

x_res2, y_res2 = LSS2.impulse_response(j=J)
b_res2 = np.array([x_res2[i][2, 0] for i in range(J)])
c_res2 = np.array([y_res2[i][0, 0] for i in range(J)])
```

```
c_res1 / σϵ, b_res1 / σϵ
```

```
(array([1.99998906e-11, 1.89473923e-11, 1.78947621e-11, 1.68421319e-11,
        1.57895017e-11]),
 array([ 0.        , -1.05263158, -1.05263158, -1.05263158, -1.05263158]))
```

```
plt.title("more informative representation")
plt.plot(range(J), c_res1 / σϵ, label="c impulse response function")
plt.plot(range(J), b_res1 / σϵ, label="b impulse response function")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa0dda359d0>
```



The above two impulse response functions show that when the consumer has the information assumed in the more informative representation (10.1), his response to receiving a positive shock of ϵ_t is to leave his consumption unchanged and to save the entire amount of his extra income and then forever roll over the extra bonds that he holds.

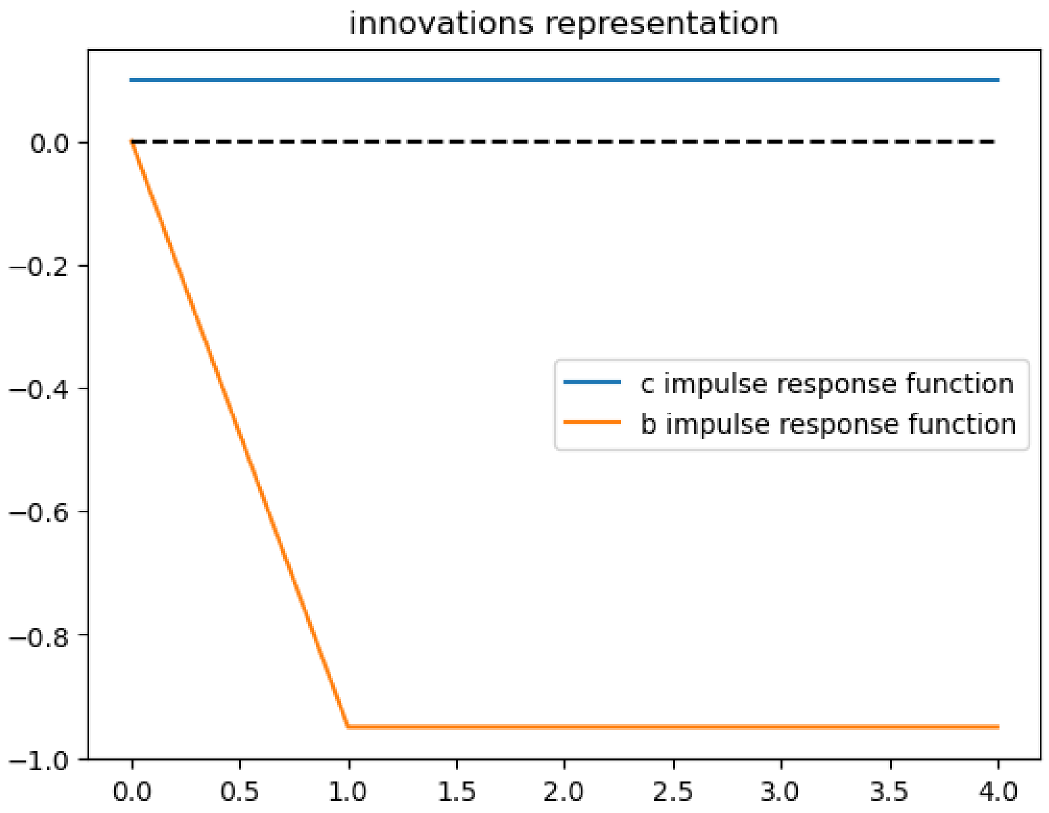
To see this notice, that starting from next period on, his debt permanently **decreases** by β^{-1}

```
c_res2 / σa, b_res2 / σa
```

```
(array([0.0975, 0.0975, 0.0975, 0.0975, 0.0975]),
 array([ 0.   , -0.95, -0.95, -0.95, -0.95]))
```

```
plt.title("innovations representation")
plt.plot(range(J), c_res2 / σa, label="c impulse response function")
plt.plot(range(J), b_res2 / σa, label="b impulse response function")
plt.plot([0, J-1], [0, 0], '--', color='k')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa0dd6a3410>
```



The above impulse responses show that when the consumer has only the information that is assumed to be available under the innovations representation (10.2) for $\{y_t - y_{t-1}\}$, he responds to a positive a_t by **permanently** increasing his consumption.

He accomplishes this by consuming a fraction $(1 - \beta^2)$ of the increment a_t to his nonfinancial income and saving the rest, thereby lowering b_{t+1} in order to finance the permanent increment in his consumption.

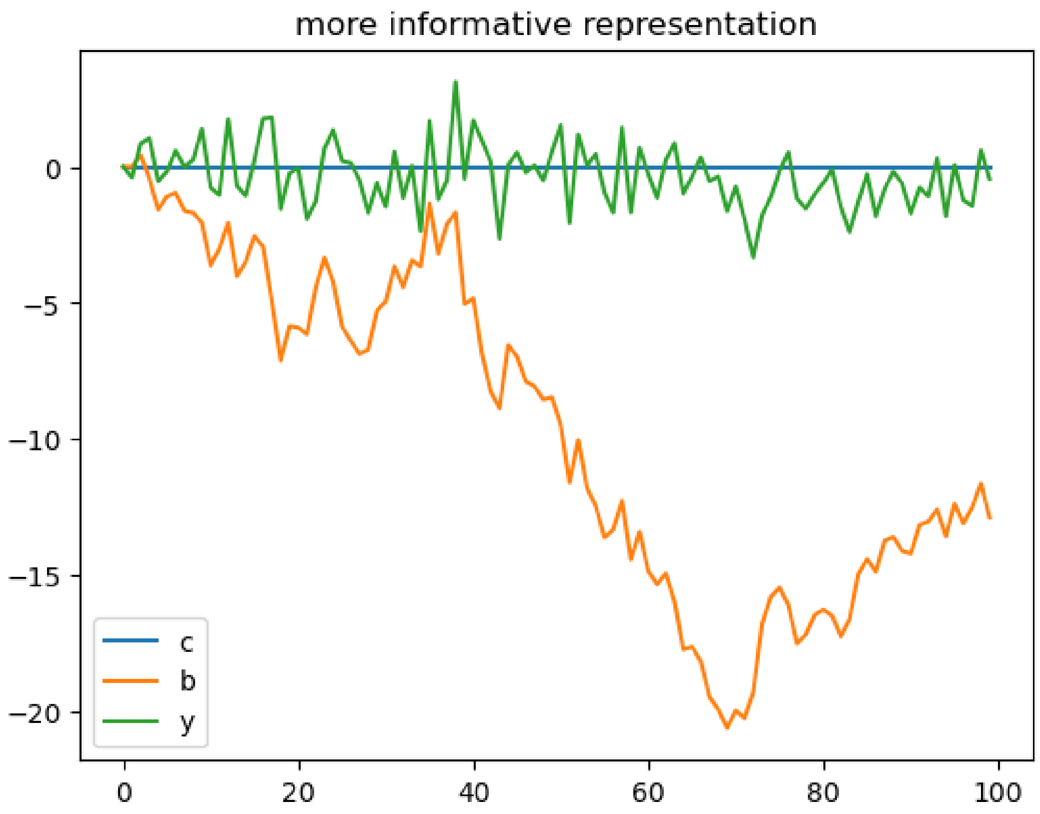
The preceding computations confirm what we had derived earlier using paper and pencil.

Now let's simulate some paths of consumption and debt for our two types of consumers while always presenting both types with the same $\{y_t\}$ path.

```
# Set time length for simulation
T = 100
```

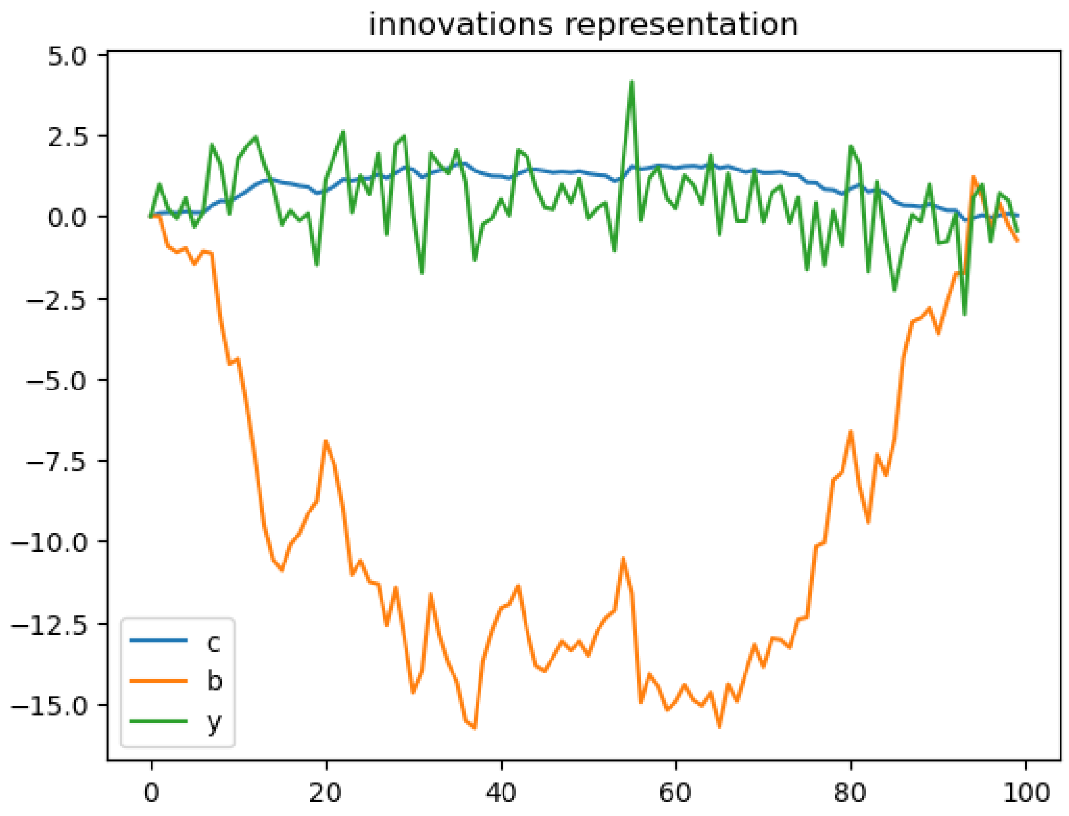
```
x1, y1 = LSS1.simulate(ts_length=T)
plt.plot(range(T), y1[0, :], label="c")
plt.plot(range(T), x1[2, :], label="b")
plt.plot(range(T), x1[0, :], label="y")
plt.title("more informative representation")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa0dd08b410>
```



```
x2, y2 = LSS2.simulate(ts_length=T)
plt.plot(range(T), y2[0, :], label="c")
plt.plot(range(T), x2[2, :], label="b")
plt.plot(range(T), x2[0, :], label="y")
plt.title("innovations representation")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa0dd223810>
```



10.10 Simulating Income Process and Two Associated Shock Processes

We now form a **single** $\{y_t\}_{t=0}^{T}$ realization that we will use to simulate decisions associated with our two types of consumer.

We accomplish this in the following steps.

1. We form a $\{y_t, \epsilon_t\}$ realization by drawing a long simulation of $\{\epsilon_t\}_{t=0}^{T}$, where T is a big integer, $\epsilon_t = \sigma_\epsilon v_t$, v_t is a standard normal scalar, $y_0 = 100$, and

$$y_{t+1} - y_t = -\beta^{-1}\epsilon_t + \epsilon_{t+1}.$$

2. We take the $\{y_t\}$ realization generated in step 1 and form an innovation process $\{a_t\}$ from the formulas

$$a_0 = 0$$
$$a_t = \sum_{j=0}^{t-1} \beta^j (y_{t-j} - y_{t-j-1}) + \beta^t a_0, \quad t \geq 1$$

3. We throw away the first S observations and form a sample $\{y_t, \epsilon_t, a_t\}_{S+1}^{T}$ as the realization that we'll use in the following steps.

4. We use the step 3 realization to **evaluate** and **simulate** the decision rules for c_t, b_t that Python has computed for us above.

The above steps implement the experiment of comparing decisions made by two consumers having **identical** incomes at each date but at each date having **different** information about their future incomes.

10.11 Calculating Innovations in Another Way

Here we use formula (10.3) above to compute a_{t+1} as a function of the history $\epsilon_{t+1}, \epsilon_t, \epsilon_{t-1}, \ldots$

Thus, we compute

$$\begin{aligned}
a_{t+1} &= \beta a_t + \epsilon_{t+1} - \beta^{-1}\epsilon_t \\
&= \beta\left(\beta a_{t-1} + \epsilon_t - \beta^{-1}\epsilon_{t-1}\right) + \epsilon_{t+1} - \beta^{-1}\epsilon_t \\
&= \beta^2 a_{t-1} + \beta\left(\epsilon_t - \beta^{-1}\epsilon_{t-1}\right) + \epsilon_{t+1} - \beta^{-1}\epsilon_t \\
&= \qquad \vdots \qquad \vdots \\
&= \beta^{t+1} a_0 + \sum_{j=0}^{t} \beta^j \left(\epsilon_{t+1-j} - \beta^{-1}\epsilon_{t-j}\right) \\
&= \beta^{t+1} a_0 + \epsilon_{t+1} + \left(\beta - \beta^{-1}\right) \sum_{j=0}^{t-1} \beta^j \epsilon_{t-j} - \beta^{t-1}\epsilon_0.
\end{aligned}$$

We can verify that we recover the same $\{a_t\}$ sequence computed earlier.

10.12 Another Invertibility Issue

This *quantecon lecture* contains another example of a shock-invertibility issue that is endemic to the LQ permanent income or consumption smoothing model.

The technical issue discussed there is ultimately the source of the shock-invertibility issues discussed by Eric Leeper, Todd Walker, and Susan Yang [Leeper *et al.*, 2013] in their analysis of **fiscal foresight**.

CHAPTER

ELEVEN

CONSUMPTION SMOOTHING WITH COMPLETE AND INCOMPLETE MARKETS

Contents

In addition to what's in Anaconda, this lecture uses the library:

```
!pip install --upgrade quantecon
```

11.1 Overview

This lecture describes two types of consumption-smoothing models.

- one is in the **complete markets** tradition of Kenneth Arrow
- the other is in the **incomplete markets** tradition of Hall [Hall, 1978]

Complete markets allow a consumer to buy and sell claims contingent on all possible states of the world.

Incomplete markets allow a consumer to buy and sell a limited set of securities, often only a single risk-free security.

Hall [Hall, 1978] worked in an incomplete markets tradition by assuming that the only asset that can be traded is a risk-free one-period bond.

Hall assumed an exogenous stochastic process of nonfinancial income and an exogenous and time-invariant gross interest rate on one-period risk-free debt that equals β^{-1}, where $\beta \in (0, 1)$ is also a consumer's intertemporal discount factor.

This is equivalent to saying that it costs β of time t consumption to buy one unit of consumption at time $t + 1$ for sure.

So β is the price of a one-period risk-free claim to consumption next period.

We preserve Hall's assumption about the interest rate when we describe an incomplete markets version of our model.

In addition, we extend Hall's assumption about the risk-free interest rate to an appropriate counterpart when we create another model in which there are markets in a complete array of one-period Arrow state-contingent securities.

We'll consider two closely related alternative assumptions about the consumer's exogenous nonfinancial income process:

- that it is generated by a finite N state Markov chain (setting $N = 2$ most of the time in this lecture)
- that it is described by a linear state space model with a continuous state vector in $\mathbb{R}^n$ driven by a Gaussian vector IID shock process

We'll spend most of this lecture studying the finite-state Markov specification, but will begin by studying the linear state space specification because it is so closely linked to earlier lectures.

Let's start with some imports:

```python
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt
import scipy.linalg as la
```

11.1.1 Relationship to Other Lectures

This lecture can be viewed as a followup to Optimal Savings II: LQ Techniques

This lecture is also a prologomenon to a lecture on tax-smoothing *Tax Smoothing with Complete and Incomplete Markets*

11.2 Background

Outcomes in consumption-smoothing models emerge from two sources:

- a consumer who wants to maximize an intertemporal objective function that expresses its preference for paths of consumption that are *smooth* in the sense of varying as little as possible both across time and across realized Markov states
- opportunities that allow the consumer to transform an erratic nonfinancial income process into a smoother consumption process by buying and selling one or more financial securities

In the **complete markets version**, each period the consumer can buy or sell a complete set of one-period ahead state-contingent securities whose payoffs depend on next period's realization of the Markov state.

- In the two-state Markov chain case, two such securities are traded each period.
- In an N state Markov state version, N such securities are traded each period.
- In a continuous state Markov state version, a continuum of such securities is traded each period.

These state-contingent securities are commonly called Arrow securities, after Kenneth Arrow.

In the **incomplete markets version**, the consumer can buy and sell only one security each period, a risk-free one-period bond with gross one-period return β^{-1}.

11.3 Linear State Space Version of Complete Markets Model

We'll study a complete markets model adapted to a setting with a continuous Markov state like that in the first lecture on the permanent income model.

In that model

- a consumer can trade only a single risk-free one-period bond bearing gross one-period risk-free interest rate equal to β^{-1}.
- a consumer's exogenous nonfinancial income is governed by a linear state space model driven by Gaussian shocks, the kind of model studied in an earlier lecture about linear state space models.

Let's write down a complete markets counterpart of that model.

Suppose that nonfinancial income is governed by the state space system

$$\begin{aligned} x_{t+1} &= Ax_t + Cw_{t+1} \\ y_t &= S_y x_t \end{aligned}$$

where x_t is an $n \times 1$ vector and $w_{t+1} \sim N(0, I)$ is IID over time.

We want a natural counterpart of the Hall assumption that the one-period risk-free gross interest rate is β^{-1}.

We make the good guess that prices of one-period ahead Arrow securities are described by the **pricing kernel**

$$q_{t+1}(x_{t+1} \mid x_t) = \beta\phi(x_{t+1} \mid Ax_t, CC') \tag{11.1}$$

where $\phi(\cdot \mid \mu, \Sigma)$ is a multivariate Gaussian distribution with mean vector μ and covariance matrix Σ.

With the pricing kernel $q_{t+1}(x_{t+1} \mid x_t)$ in hand, we can price claims to consumption at time $t+1$ consumption that pay off when $x_{t+1} \in S$ at time $t+1$:

$$\int_S q_{t+1}(x_{t+1} \mid x_t) dx_{t+1}$$

where S is a subset of $\mathbb{R}^n$.

The price $\int_S q_{t+1}(x_{t+1} \mid x_t) dx_{t+1}$ of such a claim depends on state x_t because the prices of the x_{t+1}-contingent securities depend on x_t through the pricing kernel $q(x_{t+1} \mid x_t)$.

Let $b(x_{t+1})$ be a vector of state-contingent debt due at $t+1$ as a function of the $t+1$ state x_{t+1}.

Using the pricing kernel assumed in (11.1), the value at t of $b(x_{t+1})$ is evidently

$$\beta \int b(x_{t+1})\phi(x_{t+1} \mid Ax_t, CC')dx_{t+1} = \beta\mathbb{E}_t b_{t+1}$$

In our complete markets setting, the consumer faces a sequence of budget constraints

$$c_t + b_t = y_t + \beta\mathbb{E}_t b_{t+1}, \quad t \geq 0$$

Please note that

$$\beta E_t b_{t+1} = \beta \int \phi_{t+1}(x_{t+1}|Ax_t, CC')b_{t+1}(x_{t+1})dx_{t+1}$$

or

$$\beta E_t b_{t+1} = \int q_{t+1}(x_{t+1}|x_t)b_{t+1}(x_{t+1})dx_{t+1}$$

which verifies that $\beta E_t b_{t+1}$ is the **value** of time $t+1$ state-contingent claims on time $t+1$ consumption issued by the consumer at time t

We can solve the time t budget constraint forward to obtain

$$b_t = \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j (y_{t+j} - c_{t+j})$$

The consumer cares about the expected value of

$$\sum_{t=0}^{\infty} \beta^t u(c_t), \quad 0 < \beta < 1$$

In the incomplete markets version of the model, we assumed that $u(c_t) = -(c_t - \gamma)^2$, so that the above utility functional became

$$-\sum_{t=0}^{\infty} \beta^t (c_t - \gamma)^2, \quad 0 < \beta < 1$$

But in the complete markets version, it is tractable to assume a more general utility function that satisfies $u' > 0$ and $u'' < 0$.

First-order conditions for the consumer's problem with complete markets and our assumption about Arrow securities prices are

$$u'(c_{t+1}) = u'(c_t) \quad \text{for all } t \geq 0$$

which implies $c_t = \bar{c}$ for some $\bar{c}$.

So it follows that

$$b_t = \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j (y_{t+j} - \bar{c})$$

or

$$b_t = S_y (I - \beta A)^{-1} x_t - \frac{1}{1-\beta} \bar{c} \tag{11.2}$$

where $\bar{c}$ satisfies

$$\bar{b}_0 = S_y (I - \beta A)^{-1} x_0 - \frac{1}{1-\beta} \bar{c} \tag{11.3}$$

where $\bar{b}_0$ is an initial level of the consumer's debt due at time $t = 0$, specified as a parameter of the problem.

Thus, in the complete markets version of the consumption-smoothing model, $c_t = \bar{c}, \forall t \geq 0$ is determined by (11.3) and the consumer's debt is the fixed function of the state x_t described by (11.2).

Please recall that in the LQ permanent income model studied in permanent income model, the state is x_t, b_t, where b_t is a complicated function of past state vectors x_{t-j}.

Notice that in contrast to that incomplete markets model, at time t the state vector is x_t alone in our complete markets model.

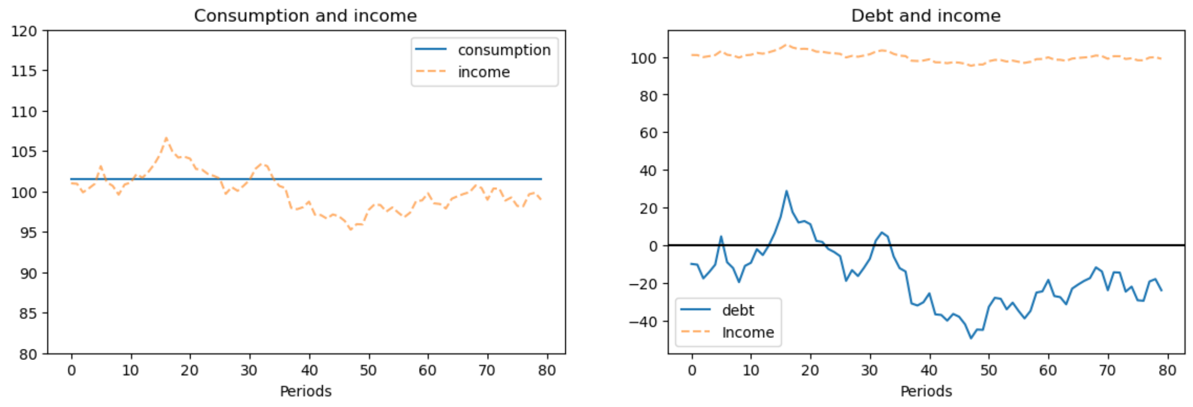
Here's an example that shows how in this setting the availability of insurance against fluctuating nonfinancial income allows the consumer completely to smooth consumption across time and across states of the world

```python
def complete_ss(β, b0, x0, A, C, S_y, T=12):
    """
    Computes the path of consumption and debt for the previously described
    complete markets model where exogenous income follows a linear
    state space
    """
    # Create a linear state space for simulation purposes
    # This adds "b" as a state to the linear state space system
    # so that setting the seed places shocks in same place for
    # both the complete and incomplete markets economy
    # Atilde = np.vstack([np.hstack([A, np.zeros((A.shape[0], 1))]),
    #                     np.zeros((1, A.shape[1] + 1))])
    # Ctilde = np.vstack([C, np.zeros((1, 1))])
    # S_ytilde = np.hstack([S_y, np.zeros((1, 1))])

    lss = qe.LinearStateSpace(A, C, S_y, mu_0=x0)

    # Add extra state to initial condition
    # x0 = np.hstack([x0, np.zeros(1)])

    # Compute the (I - β * A)^{-1}
    rm = la.inv(np.eye(A.shape[0]) - β * A)

    # Constant level of consumption
    cbar = (1 - β) * (S_y @ rm @ x0 - b0)
    c_hist = np.full(T, cbar)

    # Debt
    x_hist, y_hist = lss.simulate(T)
    b_hist = np.squeeze(S_y @ rm @ x_hist - cbar / (1 - β))


    return c_hist, b_hist, np.squeeze(y_hist), x_hist


# Define parameters
N_simul = 80
α, ρ1, ρ2 = 10.0, 0.9, 0.0
σ = 1.0

A = np.array([[1., 0., 0.],
              [α,  ρ1, ρ2],
              [0., 1., 0.]])
C = np.array([[0.], [σ], [0.]])
S_y = np.array([[1,  1.0, 0.]])
β, b0 = 0.95, -10.0
x0 = np.array([1.0, α / (1 - ρ1), α / (1 - ρ1)])

# Do simulation for complete markets
s = np.random.randint(0, 10000)
np.random.seed(s)  # Seeds get set the same for both economies
out = complete_ss(β, b0, x0, A, C, S_y, 80)
c_hist_com, b_hist_com, y_hist_com, x_hist_com = out

fig, ax = plt.subplots(1, 2, figsize=(14, 4))

# Consumption plots
```

(continues on next page)

(continued from previous page)

```python
ax[0].set_title('Consumption and income')
ax[0].plot(np.arange(N_simul), c_hist_com, label='consumption')
ax[0].plot(np.arange(N_simul), y_hist_com, label='income', alpha=.6, linestyle='--')
ax[0].legend()
ax[0].set_xlabel('Periods')
ax[0].set_ylim([80, 120])

# Debt plots
ax[1].set_title('Debt and income')
ax[1].plot(np.arange(N_simul), b_hist_com, label='debt')
ax[1].plot(np.arange(N_simul), y_hist_com, label='Income', alpha=.6, linestyle='--')
ax[1].legend()
ax[1].axhline(0, color='k')
ax[1].set_xlabel('Periods')

plt.show()
```



11.3.1 Interpretation of Graph

In the above graph, please note that:

- nonfinancial income fluctuates in a stationary manner.
- consumption is completely constant.
- the consumer's debt fluctuates in a stationary manner; in fact, in this case, because nonfinancial income is a first-order autoregressive process, the consumer's debt is an exact affine function (meaning linear plus a constant) of the consumer's nonfinancial income.

11.3.2 Incomplete Markets Version

The incomplete markets version of the model with nonfinancial income being governed by a linear state space system is described in permanent income model.

In that incomplete markerts setting, consumption follows a random walk and the consumer's debt follows a process with a unit root.

11.3.3 Finite State Markov Income Process

We now turn to a finite-state Markov version of the model in which the consumer's nonfinancial income is an exact function of a Markov state that takes one of N values.

We'll start with a setting in which in each version of our consumption-smoothing model, nonfinancial income is governed by a two-state Markov chain (it's easy to generalize this to an N state Markov chain).

In particular, the *state* $s_t \in \{1, 2\}$ follows a Markov chain with transition probability matrix

$$P_{ij} = \mathbb{P}\{s_{t+1} = j \,|\, s_t = i\}$$

where $\mathbb{P}$ means conditional probability

Nonfinancial income $\{y_t\}$ obeys

$$y_t = \begin{cases} \bar y_1 & \text{if } s_t = 1 \\ \bar y_2 & \text{if } s_t = 2 \end{cases}$$

A consumer wishes to maximize

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t u(c_t)\right] \quad \text{where} \quad u(c_t) = -(c_t - \gamma)^2 \quad \text{and} \quad 0 < \beta < 1 \tag{11.4}$$

Here $\gamma > 0$ is a bliss level of consumption

11.3.4 Market Structure

Our complete and incomplete markets models differ in how thoroughly the market structure allows a consumer to transfer resources across time and Markov states, there being more transfer opportunities in the complete markets setting than in the incomplete markets setting.

Watch how these differences in opportunities affect

- how smooth consumption is across time and Markov states
- how the consumer chooses to make his levels of indebtedness behave over time and across Markov states

11.4 Model 1 (Complete Markets)

At each date $t \geq 0$, the consumer trades a full array of **one-period ahead Arrow securities**.

We assume that prices of these securities are exogenous to the consumer.

Exogenous means that they are unaffected by the consumer's decisions.

In Markov state s_t at time t, one unit of consumption in state s_{t+1} at time $t+1$ costs $q(s_{t+1} \,|\, s_t)$ units of the time t consumption good.

The prices $q(s_{t+1} \,|\, s_t)$ are given and can be organized into a matrix Q with $Q_{ij} = q(j|i)$

At time $t = 0$, the consumer starts with an inherited level of debt due at time 0 of b_0 units of time 0 consumption goods.

The consumer's budget constraint at $t \geq 0$ in Markov state s_t is

$$c_t + b_t \leq y(s_t) + \sum_j q(j \,|\, s_t)\, b_{t+1}(j \,|\, s_t) \tag{11.5}$$

where b_t is the consumer's one-period debt that falls due at time t and $b_{t+1}(j \mid s_t)$ are the consumer's time t sales of the time $t+1$ consumption good in Markov state j.

Thus

- $q(j \mid s_t) b_{t+1}(j \mid s_t)$ is a source of time t **financial income** for the consumer in Markov state s_t
- $b_t \equiv b_t(j \mid s_{t-1})$ is a source of time t **expenditures** for the consumer when $s_t = j$

Remark: We are ignoring an important technicality here, namely, that the consumer's choice of $b_{t+1}(j| \, s_t)$ must respect so-called *natural debt limits* that assure that it is feasible for the consumer to repay debts due even if he consumers zero forevermore. We shall discuss such debt limits in another lecture.

A natural analog of Hall's assumption that the one-period risk-free gross interest rate is β^{-1} is

$$q(j \mid i) = \beta P_{ij} \tag{11.6}$$

To understand how this is a natural analogue, observe that in state i it costs $\sum_j q(j \mid i)$ to purchase one unit of consumption next period *for sure*, i.e., meaning no matter what Markov state j occurs at $t+1$.

Hence the **implied price** of a risk-free claim on one unit of consumption next period is

$$\sum_j q(j \mid i) = \sum_j \beta P_{ij} = \beta$$

This confirms the sense in which (11.6) is a natural counterpart to Hall's assumption that the risk-free one-period gross interest rate is $R = \beta^{-1}$.

It is timely please to recall that the gross one-period risk-free interest rate is the reciprocal of the price at time t of a risk-free claim on one unit of consumption tomorrow.

First-order necessary conditions for maximizing the consumer's expected utility subject to the sequence of budget constraints (11.5) are

$$\beta \frac{u'(c_{t+1})}{u'(c_t)} \mathbb{P}\{s_{t+1} \mid s_t\} = q(s_{t+1} \mid s_t)$$

for all s_t, s_{t+1} or, under our assumption (11.6) about Arrow security prices,

$$c_{t+1} = c_t \tag{11.7}$$

Thus, our consumer sets $c_t = \bar{c}$ for all $t \geq 0$ for some value $\bar{c}$ that it is our job now to determine along with values for $b_{t+1}(j|s_t = i)$ for $i = 1, 2$ and $j = 1, 2$.

We'll use a *guess and verify* method to determine these objects

Guess: We'll make the plausible guess that

$$b_{t+1}(s_{t+1} = j \mid s_t = i) = b(j), \quad i = 1, 2; \;\; j = 1, 2 \tag{11.8}$$

so that the amount borrowed today depends only on *tomorrow's* Markov state. (Why is this is a plausible guess?)

To determine $\bar{c}$, we shall deduce implications of the consumer's budget constraints in each Markov state today and our guess (11.8) about the consumer's debt level choices.

For $t \geq 1$, these imply

$$\begin{aligned} \bar{c} + b(1) &= y(1) + q(1 \mid 1) b(1) + q(2 \mid 1) b(2) \\ \bar{c} + b(2) &= y(2) + q(1 \mid 2) b(1) + q(2 \mid 2) b(2) \end{aligned} \tag{11.9}$$

or

$$\begin{bmatrix} b(1) \\ b(2) \end{bmatrix} + \begin{bmatrix} \bar{c} \\ \bar{c} \end{bmatrix} = \begin{bmatrix} y(1) \\ y(2) \end{bmatrix} + \beta \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \begin{bmatrix} b(1) \\ b(2) \end{bmatrix}$$

These are 2 equations in the 3 unknowns $\bar{c}, b(1), b(2)$

To get a third equation, we assume that at time $t = 0$, b_0 is debt due; and we assume that at time $t = 0$, the Markov state $s_0 = 1$

(We could instead have assumed that at time $t = 0$ the Markov state $s_0 = 2$, which would affect our answer as we shall see)

Since we have assumed that $s_0 = 1$, the budget constraint at time $t = 0$ is

$$\bar{c} + b_0 = y(1) + q(1 \,|\, 1)b(1) + q(2 \,|\, 1)b(2) \tag{11.10}$$

where b_0 is the (exogenous) debt the consumer is assumed to bring into period 0

If we substitute (11.10) into the first equation of (11.9) and rearrange, we discover that

$$b(1) = b_0 \tag{11.11}$$

We can then use the second equation of (11.9) to deduce the restriction

$$y(1) - y(2) + [q(1 \,|\, 1) - q(1 \,|\, 2) - 1]b_0 + [q(2 \,|\, 1) + 1 - q(2 \,|\, 2)]b(2) = 0, \tag{11.12}$$

an equation that we can solve for the unknown $b(2)$.

Knowing $b(1)$ and $b(2)$, we can solve equation (11.10) for the constant level of consumption $\bar{c}$.

11.4.1 Key Outcomes

The preceding calculations indicate that in the complete markets version of our model, we obtain the following striking results:

- The consumer chooses to make consumption perfectly constant across time and across Markov states.
- State-contingent debt purchases $b_{t+1}(s_{t+1} = j | s_t = i)$ depend only on j
- If the initial Markov state is $s_0 = j$ and initial consumer debt is b_0, then debt in Markov state j satisfies $b(j) = b_0$

To summarize what we have achieved up to now, we have computed the constant level of consumption $\bar{c}$ and indicated how that level depends on the underlying specifications of preferences, Arrow securities prices, the stochastic process of exogenous nonfinancial income, and the initial debt level b_0

- The consumer's debt neither accumulates, nor decumulates, nor drifts – instead, the debt level each period is an exact function of the Markov state, so in the two-state Markov case, it switches between two values.
- We have verified guess (11.8).
- When the state s_t returns to the initial state s_0, debt returns to the initial debt level.
- Debt levels in all other states depend on virtually all remaining parameters of the model.

11.4.2 Code

Here's some code that, among other things, contains a function called `consumption_complete()`.

This function computes $\{b(i)\}_{i=1}^N, \bar{c}$ as outcomes given a set of parameters for the general case with N Markov states under the assumption of complete markets

```
class ConsumptionProblem:
    """
    The data for a consumption problem, including some default values.
    """

    def __init__(self,
                 β=.96,
                 y=[2, 1.5],
                 b0=3,
                 P=[[.8, .2],
                    [.4, .6]],
                 init=0):
        """
        Parameters
        ----------

        β : discount factor
        y : list containing the two income levels
        b0 : debt in period 0 (= initial state debt level)
        P : 2x2 transition matrix
        init : index of initial state s0
        """
        self.β = β
        self.y = np.asarray(y)
        self.b0 = b0
        self.P = np.asarray(P)
        self.init = init

    def simulate(self, N_simul=80, random_state=1):
        """
        Parameters
        ----------

        N_simul : number of periods for simulation
        random_state : random state for simulating Markov chain
        """
        # For the simulation define a quantecon MC class
        mc = qe.MarkovChain(self.P)
        s_path = mc.simulate(N_simul, init=self.init, random_state=random_state)

        return s_path


def consumption_complete(cp):
    """
    Computes endogenous values for the complete market case.

    Parameters
    ----------

    cp : instance of ConsumptionProblem

    Returns
    -------

        c_bar : constant consumption
        b : optimal debt in each state
```

(continues on next page)

(continued from previous page)

```
    associated with the price system

        Q = β * P
    """
    β, P, y, b0, init = cp.β, cp.P, cp.y, cp.b0, cp.init   # Unpack

    Q = β * P                              # assumed price system

    # construct matrices of augmented equation system
    n = P.shape[0] + 1

    y_aug = np.empty((n, 1))
    y_aug[0, 0] = y[init] - b0
    y_aug[1:, 0] = y

    Q_aug = np.zeros((n, n))
    Q_aug[0, 1:] = Q[init, :]
    Q_aug[1:, 1:] = Q

    A = np.zeros((n, n))
    A[:, 0] = 1
    A[1:, 1:] = np.eye(n-1)

    x = np.linalg.inv(A - Q_aug) @ y_aug

    c_bar = x[0, 0]
    b = x[1:, 0]

    return c_bar, b


def consumption_incomplete(cp, s_path):
    """
    Computes endogenous values for the incomplete market case.

    Parameters
    ----------

    cp : instance of ConsumptionProblem
    s_path : the path of states
    """
    β, P, y, b0 = cp.β, cp.P, cp.y, cp.b0  # Unpack

    N_simul = len(s_path)

    # Useful variables
    n = len(y)
    y.shape = (n, 1)
    v = np.linalg.inv(np.eye(n) - β * P) @ y

    # Store consumption and debt path
    b_path, c_path = np.ones(N_simul+1), np.ones(N_simul)
    b_path[0] = b0

    # Optimal decisions from (12) and (13)
```

(continues on next page)

(continued from previous page)

```
    db = ((1 - β) * v - y) / β

    for i, s in enumerate(s_path):
        c_path[i] = (1 - β) * (v - np.full((n, 1), b_path[i]))[s, 0]
        b_path[i + 1] = b_path[i] + db[s, 0]

    return c_path, b_path[:-1], y[s_path]
```

Let's test by checking that $\bar{c}$ and b_2 satisfy the budget constraint

```
cp = ConsumptionProblem()
c_bar, b = consumption_complete(cp)
np.isclose(c_bar + b[1] - cp.y[1] - (cp.β * cp.P)[1, :] @ b, 0)
```

```
True
```

Below, we'll take the outcomes produced by this code – in particular the implied consumption and debt paths – and compare them with outcomes from an incomplete markets model in the spirit of Hall [Hall, 1978]

11.5 Model 2 (One-Period Risk-Free Debt Only)

This is a version of the original model of Hall (1978) in which the consumer's ability to substitute intertemporally is constrained by his ability to buy or sell only one security, a risk-free one-period bond bearing a constant gross interest rate that equals β^{-1}.

Given an initial debt b_0 at time 0, the consumer faces a sequence of budget constraints

$$c_t + b_t = y_t + \beta b_{t+1}, \quad t \geq 0$$

where β is the price at time t of a risk-free claim on one unit of time consumption at time $t+1$.

First-order conditions for the consumer's problem are

$$\sum_j u'(c_{t+1,j}) P_{ij} = u'(c_{t,i})$$

For our assumed quadratic utility function this implies

$$\sum_j c_{t+1,j} P_{ij} = c_{t,i} \tag{11.13}$$

which for our finite-state Markov setting is Hall's (1978) conclusion that consumption follows a random walk.

As we saw in our first lecture on the permanent income model, this leads to

$$b_t = \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} - (1-\beta)^{-1} c_t \tag{11.14}$$

and

$$c_t = (1-\beta) \left[\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} - b_t \right] \tag{11.15}$$

Equation (11.15) expresses c_t as a net interest rate factor $1-\beta$ times the sum of the expected present value of nonfinancial income $\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j}$ and financial wealth $-b_t$.

Substituting (11.15) into the one-period budget constraint and rearranging leads to

$$b_{t+1} - b_t = \beta^{-1} \left[(1-\beta)\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} - y_t \right] \tag{11.16}$$

Now let's calculate the key term $\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j}$ in our finite Markov chain setting.

Define the expected discounted present value of non-financial income

$$v_t := \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j}$$

which in the spirit of dynamic programming we can write as a *Bellman equation*

$$v_t := y_t + \beta \mathbb{E}_t v_{t+1}$$

In our two-state Markov chain setting, $v_t = v(1)$ when $s_t = 1$ and $v_t = v(2)$ when $s_t = 2$.

Therefore, we can write our Bellman equation as

$$\begin{aligned} v(1) &= y(1) + \beta P_{11} v(1) + \beta P_{12} v(2) \\ v(2) &= y(2) + \beta P_{21} v(1) + \beta P_{22} v(2) \end{aligned}$$

or

$$\vec{v} = \vec{y} + \beta P \vec{v}$$

where $\vec{v} = \begin{bmatrix} v(1) \\ v(2) \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} y(1) \\ y(2) \end{bmatrix}$.

We can also write the last expression as

$$\vec{v} = (I - \beta P)^{-1} \vec{y}$$

In our finite Markov chain setting, from expression (11.15), consumption at date t when debt is b_t and the Markov state today is $s_t = i$ is evidently

$$c(b_t, i) = (1-\beta) \left([(I - \beta P)^{-1} \vec{y}]_i - b_t \right) \tag{11.17}$$

and the increment to debt is

$$b_{t+1} - b_t = \beta^{-1} [(1-\beta) v(i) - y(i)] \tag{11.18}$$

11.5.1 Summary of Outcomes

In contrast to outcomes in the complete markets model, in the incomplete markets model

- consumption drifts over time as a random walk; the level of consumption at time t depends on the level of debt that the consumer brings into the period as well as the expected discounted present value of nonfinancial income at t.
- the consumer's debt drifts upward over time in response to low realizations of nonfinancial income and drifts downward over time in response to high realizations of nonfinancial income.
- the drift over time in the consumer's debt and the dependence of current consumption on today's debt level account for the drift over time in consumption.

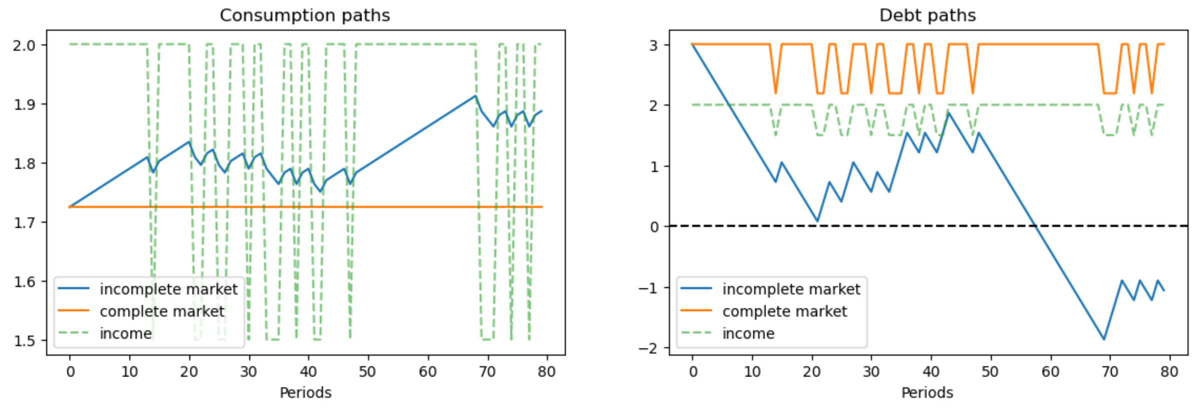
11.5.2 The Incomplete Markets Model

The code above also contains a function called `consumption_incomplete()` that uses (11.17) and (11.18) to

- simulate paths of y_t, c_t, b_{t+1}
- plot these against values of $\bar{c}, b(s_1), b(s_2)$ found in a corresponding complete markets economy

Let's try this, using the same parameters in both complete and incomplete markets economies

```
cp = ConsumptionProblem()
s_path = cp.simulate()
N_simul = len(s_path)

c_bar, debt_complete = consumption_complete(cp)

c_path, debt_path, y_path = consumption_incomplete(cp, s_path)

fig, ax = plt.subplots(1, 2, figsize=(14, 4))

ax[0].set_title('Consumption paths')
ax[0].plot(np.arange(N_simul), c_path, label='incomplete market')
ax[0].plot(np.arange(N_simul), np.full(N_simul, c_bar),
                    label='complete market')
ax[0].plot(np.arange(N_simul), y_path, label='income', alpha=.6, ls='--')
ax[0].legend()
ax[0].set_xlabel('Periods')

ax[1].set_title('Debt paths')
ax[1].plot(np.arange(N_simul), debt_path, label='incomplete market')
ax[1].plot(np.arange(N_simul), debt_complete[s_path],
           label='complete market')
ax[1].plot(np.arange(N_simul), y_path, label='income', alpha=.6, ls='--')
ax[1].legend()
ax[1].axhline(0, color='k', ls='--')
ax[1].set_xlabel('Periods')

plt.show()
```



In the graph on the left, for the same sample path of nonfinancial income y_t, notice that

- consumption is constant when there are complete markets, but takes a random walk in the incomplete markets version of the model.

- the consumer's debt oscillates between two values that are functions of the Markov state in the complete markets model, while the consumer's debt drifts in a "unit root" fashion in the incomplete markets economy.

11.5.3 A sequel

In *tax smoothing with complete and incomplete markets*, we reinterpret the mathematics and Python code presented in this lecture in order to construct tax-smoothing models in the incomplete markets tradition of Barro [Barro, 1979] as well as in the complete markets tradition of Lucas and Stokey [Lucas and Stokey, 1983].

CHAPTER

TWELVE

TAX SMOOTHING WITH COMPLETE AND INCOMPLETE MARKETS

Contents

In addition to what's in Anaconda, this lecture uses the library:

```
!pip install --upgrade quantecon
```

12.1 Overview

This lecture describes tax-smoothing models that are counterparts to consumption-smoothing models in *Consumption Smoothing with Complete and Incomplete Markets*.

- one is in the **complete markets** tradition of Lucas and Stokey [Lucas and Stokey, 1983].
- the other is in the **incomplete markets** tradition of Barro [Barro, 1979].

Complete markets allow a government to buy or sell claims contingent on all possible Markov states.

Incomplete markets allow a government to buy or sell only a limited set of securities, often only a single risk-free security.

Barro [Barro, 1979] worked in an incomplete markets tradition by assuming that the only asset that can be traded is a risk-free one period bond.

In his consumption-smoothing model, Hall [Hall, 1978] had assumed an exogenous stochastic process of nonfinancial income and an exogenous gross interest rate on one period risk-free debt that equals β^{-1}, where $\beta \in (0, 1)$ is also a consumer's intertemporal discount factor.

Barro [Barro, 1979] made an analogous assumption about the risk-free interest rate in a tax-smoothing model that turns out to have the same mathematical structure as Hall's consumption-smoothing model.

To get Barro's model from Hall's, all we have to do is to rename variables.

We maintain Hall's and Barro's assumption about the interest rate when we describe an incomplete markets version of our model.

In addition, we extend their assumption about the interest rate to an appropriate counterpart to create a "complete markets" model in the style of Lucas and Stokey [Lucas and Stokey, 1983].

12.1.1 Isomorphism between Consumption and Tax Smoothing

For each version of a consumption-smoothing model, a tax-smoothing counterpart can be obtained simply by relabeling

- consumption as tax collections
- a consumer's one-period utility function as a government's one-period loss function from collecting taxes that impose deadweight welfare losses
- a consumer's nonfinancial income as a government's purchases
- a consumer's *debt* as a government's *assets*

Thus, we can convert the consumption-smoothing models in lecture *Consumption Smoothing with Complete and Incomplete Markets* into tax-smoothing models by setting $c_t = T_t$, $y_t = G_t$, and $-b_t = a_t$, where T_t is total tax collections, $\{G_t\}$ is an exogenous government expenditures process, and a_t is the government's holdings of one-period risk-free bonds coming maturing at the due at the beginning of time t.

For elaborations on this theme, please see Optimal Savings II: LQ Techniques and later parts of this lecture.

We'll spend most of this lecture studying acquire finite-state Markov specification, but will also treat the linear state space specification.

Link to History

For those who love history, President Thomas Jefferson's Secretary of Treasury Albert Gallatin (1807) [Gallatin, 1837] seems to have prescribed policies that come from Barro's model [Barro, 1979]

Let's start with some standard imports:

```python
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt
```

To exploit the isomorphism between consumption-smoothing and tax-smoothing models, we simply use code from *Consumption Smoothing with Complete and Incomplete Markets*

12.1.2 Code

Among other things, this code contains a function called `consumption_complete()`.

This function computes $\{b(i)\}_{i=1}^N, \bar{c}$ as outcomes given a set of parameters for the general case with N Markov states under the assumption of complete markets

```python
class ConsumptionProblem:
    """
    The data for a consumption problem, including some default values.
    """

    def __init__(self,
                 β=.96,
                 y=[2, 1.5],
                 b0=3,
```

(continues on next page)

(continued from previous page)

```
                 P=[[.8, .2],
                    [.4, .6]],
                 init=0):
        """
        Parameters
        ----------

        β : discount factor
        y : list containing the two income levels
        b0 : debt in period 0 (= initial state debt level)
        P : 2x2 transition matrix
        init : index of initial state s0
        """
        self.β = β
        self.y = np.asarray(y)
        self.b0 = b0
        self.P = np.asarray(P)
        self.init = init

    def simulate(self, N_simul=80, random_state=1):
        """
        Parameters
        ----------

        N_simul : number of periods for simulation
        random_state : random state for simulating Markov chain
        """
        # For the simulation define a quantecon MC class
        mc = qe.MarkovChain(self.P)
        s_path = mc.simulate(N_simul, init=self.init, random_state=random_state)

        return s_path


def consumption_complete(cp):
    """
    Computes endogenous values for the complete market case.

    Parameters
    ----------

    cp : instance of ConsumptionProblem

    Returns
    -------

        c_bar : constant consumption
        b : optimal debt in each state

    associated with the price system

        Q = β * P
    """
    β, P, y, b0, init = cp.β, cp.P, cp.y, cp.b0, cp.init   # Unpack

    Q = β * P                                     # assumed price system
```

(continues on next page)

(continued from previous page)

```
    # construct matrices of augmented equation system
    n = P.shape[0] + 1

    y_aug = np.empty((n, 1))
    y_aug[0, 0] = y[init] - b0
    y_aug[1:, 0] = y

    Q_aug = np.zeros((n, n))
    Q_aug[0, 1:] = Q[init, :]
    Q_aug[1:, 1:] = Q

    A = np.zeros((n, n))
    A[:, 0] = 1
    A[1:, 1:] = np.eye(n-1)

    x = np.linalg.inv(A - Q_aug) @ y_aug

    c_bar = x[0, 0]
    b = x[1:, 0]

    return c_bar, b


def consumption_incomplete(cp, s_path):
    """
    Computes endogenous values for the incomplete market case.

    Parameters
    ----------

    cp : instance of ConsumptionProblem
    s_path : the path of states
    """
    β, P, y, b0 = cp.β, cp.P, cp.y, cp.b0  # Unpack

    N_simul = len(s_path)

    # Useful variables
    n = len(y)
    y.shape = (n, 1)
    v = np.linalg.inv(np.eye(n) - β * P) @ y

    # Store consumption and debt path
    b_path, c_path = np.ones(N_simul+1), np.ones(N_simul)
    b_path[0] = b0

    # Optimal decisions from (12) and (13)
    db = ((1 - β) * v - y) / β

    for i, s in enumerate(s_path):
        c_path[i] = (1 - β) * (v - np.full((n, 1), b_path[i]))[s, 0]
        b_path[i + 1] = b_path[i] + db[s, 0]

    return c_path, b_path[:-1], y[s_path]
```

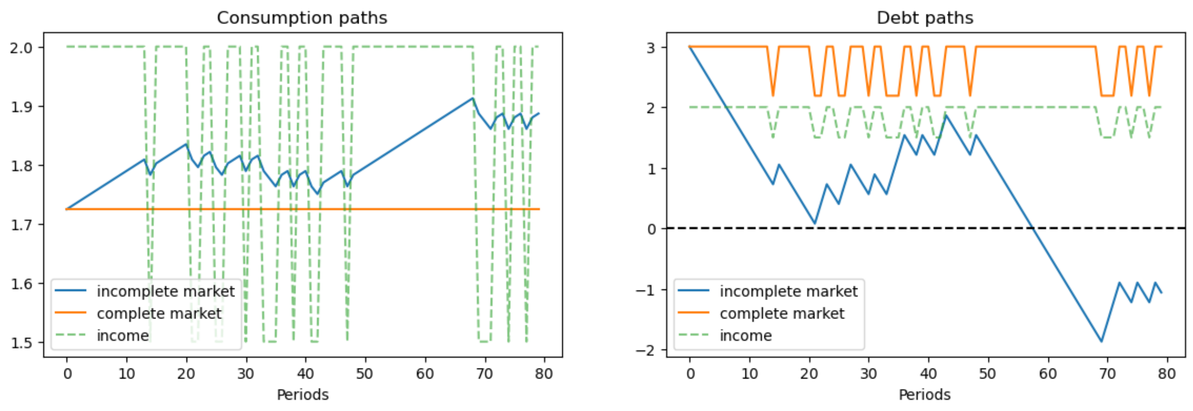
12.1.3 Revisiting the consumption-smoothing model

The code above also contains a function called `consumption_incomplete()` that uses (11.17) and (11.18) to

- simulate paths of y_t, c_t, b_{t+1}
- plot these against values of $\bar{c}, b(s_1), b(s_2)$ found in a corresponding complete markets economy

Let's try this, using the same parameters in both complete and incomplete markets economies

```python
cp = ConsumptionProblem()
s_path = cp.simulate()
N_simul = len(s_path)

c_bar, debt_complete = consumption_complete(cp)

c_path, debt_path, y_path = consumption_incomplete(cp, s_path)

fig, ax = plt.subplots(1, 2, figsize=(14, 4))

ax[0].set_title('Consumption paths')
ax[0].plot(np.arange(N_simul), c_path, label='incomplete market')
ax[0].plot(np.arange(N_simul), np.full(N_simul, c_bar), label='complete market')
ax[0].plot(np.arange(N_simul), y_path, label='income', alpha=.6, ls='--')
ax[0].legend()
ax[0].set_xlabel('Periods')

ax[1].set_title('Debt paths')
ax[1].plot(np.arange(N_simul), debt_path, label='incomplete market')
ax[1].plot(np.arange(N_simul), debt_complete[s_path], label='complete market')
ax[1].plot(np.arange(N_simul), y_path, label='income', alpha=.6, ls='--')
ax[1].legend()
ax[1].axhline(0, color='k', ls='--')
ax[1].set_xlabel('Periods')

plt.show()
```



In the graph on the left, for the same sample path of nonfinancial income y_t, notice that

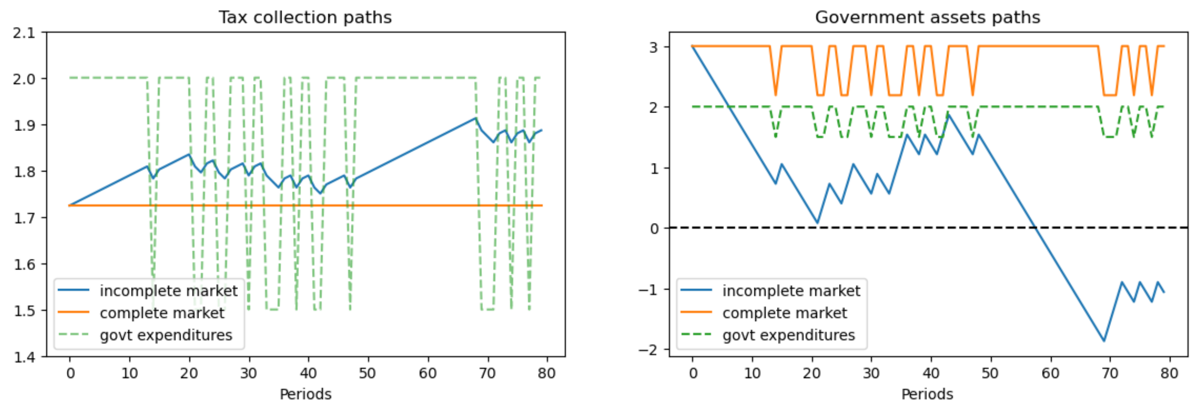
- consumption is constant when there are complete markets.
- consumption takes a random walk in the incomplete markets version of the model.
- the consumer's debt oscillates between two values that are functions of the Markov state in the complete markets model.

- the consumer's debt drifts because it contains a unit root in the incomplete markets economy.

Relabeling variables to create tax-smoothing models

As indicated above, we relabel variables to acquire tax-smoothing interpretations of the complete markets and incomplete markets consumption-smoothing models.

```
fig, ax = plt.subplots(1, 2, figsize=(14, 4))

ax[0].set_title('Tax collection paths')
ax[0].plot(np.arange(N_simul), c_path, label='incomplete market')
ax[0].plot(np.arange(N_simul), np.full(N_simul, c_bar), label='complete market')
ax[0].plot(np.arange(N_simul), y_path, label='govt expenditures', alpha=.6, ls='--')
ax[0].legend()
ax[0].set_xlabel('Periods')
ax[0].set_ylim([1.4, 2.1])

ax[1].set_title('Government assets paths')
ax[1].plot(np.arange(N_simul), debt_path, label='incomplete market')
ax[1].plot(np.arange(N_simul), debt_complete[s_path], label='complete market')
ax[1].plot(np.arange(N_simul), y_path, label='govt expenditures', ls='--')
ax[1].legend()
ax[1].axhline(0, color='k', ls='--')
ax[1].set_xlabel('Periods')

plt.show()
```



12.2 Tax Smoothing with Complete Markets

It is instructive to focus on a simple tax-smoothing example with complete markets.

This example illustrates how, in a complete markets model like that of Lucas and Stokey [Lucas and Stokey, 1983], the government purchases insurance from the private sector.

Payouts from the insurance it had purchased allows the government to avoid raising taxes when emergencies make government expenditures surge.

We assume that government expenditures take one of two values $G_1 < G_2$, where Markov state 1 means "peace" and Markov state 2 means "war".

The government budget constraint in Markov state i is

$$T_i + b_i = G_i + \sum_j Q_{ij} b_j$$

where

$$Q_{ij} = \beta P_{ij}$$

is the price today of one unit of goods in Markov state j tomorrow when the Markov state is i today.

b_i is the government's level of *assets* when it arrives in Markov state i.

That is, b_i equals one-period state-contingent claims *owed to the government* that fall due at time t when the Markov state is i.

Thus, if $b_i < 0$, it means the government **is owed** b_i or **owes** $-b_i$ when the economy arrives in Markov state i at time t.

In our examples below, this happens when in a previous war-time period the government has sold an Arrow securities paying off $-b_i$ in peacetime Markov state i

It can be enlightening to express the government's budget constraint in Markov state i as

$$T_i = G_i + \left(\sum_j Q_{ij} b_j - b_i \right)$$

in which the term $(\sum_j Q_{ij} b_j - b_i)$ equals the net amount that the government spends to purchase one-period Arrow securities that will pay off next period in Markov states $j = 1, \ldots, N$ after it has received payments b_i this period.

12.3 Returns on State-Contingent Debt

Notice that $\sum_{j'=1}^N Q_{ij'} b(j')$ is the amount that the government spends in Markov state i at time t to purchase one-period state-contingent claims that will pay off in Markov state j' at time $t + 1$.

Then the *ex post* one-period gross return on the portfolio of government assets held from state i at time t to state j at time $t + 1$ is

$$R(j|i) = \frac{b(j)}{\sum_{j'=1}^N Q_{ij'} b(j')}$$

The cumulative return earned from putting 1 unit of time t goods into the government portfolio of state-contingent securities at time t and then rolling over the proceeds into the government portfolio each period thereafter is

$$R^T(s_{t+T}, s_{t+T-1}, \ldots, s_t) \equiv R(s_{t+1}|s_t) R(s_{t+2}|s_{t+1}) \cdots R(s_{t+T}|s_{t+T-1})$$

Here is some code that computes one-period and cumulative returns on the government portfolio in the finite-state Markov version of our complete markets model.

Convention: In this code, when $P_{ij} = 0$, we arbitrarily set $R(j|i)$ to be 0.

```python
def ex_post_gross_return(b, cp):
    """
    calculate the ex post one-period gross return on the portfolio
    of government assets, given b and Q.
    """
    Q = cp.β * cp.P
```

(continues on next page)

(continued from previous page)

```
    values = Q @ b

    n = len(b)
    R = np.zeros((n, n))

    for i in range(n):
        ind = cp.P[i, :] != 0
        R[i, ind] = b[ind] / values[i]

    return R

def cumulative_return(s_path, R):
    """
    compute cumulative return from holding 1 unit market portfolio
    of government bonds, given some simulated state path.
    """
    T = len(s_path)

    RT_path = np.empty(T)
    RT_path[0] = 1
    RT_path[1:] = np.cumprod([R[s_path[t], s_path[t+1]] for t in range(T-1)])

    return RT_path
```

12.3.1 An Example of Tax Smoothing

We'll study a tax-smoothing model with two Markov states.

In Markov state 1, there is peace and government expenditures are low.

In Markov state 2, there is war and government expenditures are high.

We'll compute optimal policies in both complete and incomplete markets settings.

Then we'll feed in a **particular** assumed path of Markov states and study outcomes.

- We'll assume that the initial Markov state is state 1, which means we start from a state of peace.
- The government then experiences 3 time periods of war and come back to peace again.
- The history of Markov states is therefore $\{peace, war, war, war, peace\}$.

In addition, as indicated above, to simplify our example, we'll set the government's initial asset level to 1, so that $b_1 = 1$.

Here's code that itinitializes government assets to be unity in an initial peace time Markov state.

```
# Parameters
β = .96

# change notation y to g in the tax-smoothing example
g = [1, 2]
b0 = 1
P = np.array([[.8, .2],
              [.4, .6]])

cp = ConsumptionProblem(β, g, b0, P)
```

(continues on next page)

(continued from previous page)

```
Q = β * P

# change notation c_bar to T_bar in the tax-smoothing example
T_bar, b = consumption_complete(cp)
R = ex_post_gross_return(b, cp)
s_path = [0, 1, 1, 1, 0]
RT_path = cumulative_return(s_path, R)

print(f"P \n {P}")
print(f"Q \n {Q}")
print(f"Govt expenditures in peace and war = {g}")
print(f"Constant tax collections = {T_bar}")
print(f"Govt debts in two states = {-b}")

msg = """
Now let's check the government's budget constraint in peace and war.
Our assumptions imply that the government always purchases 0 units of the
Arrow peace security.
"""
print(msg)

AS1 = Q[0, :] @ b
# spending on Arrow security
# since the spending on Arrow peace security is not 0 anymore after we change b0 to 1
print(f"Spending on Arrow security in peace = {AS1}")
AS2 = Q[1, :] @ b
print(f"Spending on Arrow security in war = {AS2}")

print("")
# tax collections minus debt levels
print("Government tax collections minus debt levels in peace and war")
TB1 = T_bar + b[0]
print(f"T+b in peace = {TB1}")
TB2 = T_bar + b[1]
print(f"T+b in war = {TB2}")

print("")
print("Total government spending in peace and war")
G1 = g[0] + AS1
G2 = g[1] + AS2
print(f"Peace = {G1}")
print(f"War = {G2}")

print("")
print("Let's see ex-post and ex-ante returns on Arrow securities")

Π = np.reciprocal(Q)
exret = Π
print(f"Ex-post returns to purchase of Arrow securities = \n {exret}")
exant = Π * P
print(f"Ex-ante returns to purchase of Arrow securities \n {exant}")

print("")
print("The Ex-post one-period gross return on the portfolio of government assets")
print(R)
```

(continues on next page)

(continued from previous page)

```
print("")
print("The cumulative return earned from holding 1 unit market portfolio of␣
↪government bonds")
print(RT_path[-1])
```

```
P
 [[0.8 0.2]
 [0.4 0.6]]
Q
 [[0.768 0.192]
 [0.384 0.576]]
Govt expenditures in peace and war = [1, 2]
Constant tax collections = 1.2716883116883118
Govt debts in two states = [-1.         -2.62337662]

Now let's check the government's budget constraint in peace and war.
Our assumptions imply that the government always purchases 0 units of the
Arrow peace security.

Spending on Arrow security in peace = 1.2716883116883118
Spending on Arrow security in war = 1.895064935064935

Government tax collections minus debt levels in peace and war
T+b in peace = 2.2716883116883118
T+b in war = 3.895064935064935

Total government spending in peace and war
Peace = 2.2716883116883118
War = 3.895064935064935

Let's see ex-post and ex-ante returns on Arrow securities
Ex-post returns to purchase of Arrow securities =
 [[1.30208333 5.20833333]
 [2.60416667 1.73611111]]
Ex-ante returns to purchase of Arrow securities
 [[1.04166667 1.04166667]
 [1.04166667 1.04166667]]

The Ex-post one-period gross return on the portfolio of government assets
[[0.78635621 2.0629085 ]
 [0.5276864  1.38432018]]

The cumulative return earned from holding 1 unit market portfolio of government␣
 ↪bonds
2.0860704239993675
```

12.3.2 Explanation

In this example, the government always purchase 1 units of the Arrow security that pays off in peace time (Markov state 1).

And it purchases a higher amount of the security that pays off in war time (Markov state 2).

Thus, this is an example in which

- during peacetime, the government purchases *insurance* against the possibility that war breaks out next period
- during wartime, the government purchases *insurance* against the possibility that war continues another period
- so long as peace continues, the ex post return on insurance against war is low
- when war breaks out or continues, the ex post return on insurance against war is high
- given the history of states that we assumed, the value of one unit of the portfolio of government assets eventually doubles in the end because of high returns during wartime.

We recommend plugging the quantities computed above into the government budget constraints in the two Markov states and staring.

Exercise 12.3.1

Try changing the Markov transition matrix so that

$$P = \begin{bmatrix} 1 & 0 \\ .2 & .8 \end{bmatrix}$$

Also, start the system in Markov state 2 (war) with initial government assets -10, so that the government starts the war in debt and $b_2 = -10$.

12.4 More Finite Markov Chain Tax-Smoothing Examples

To interpret some episodes in the fiscal history of the United States, we find it interesting to study a few more examples.

We compute examples in an N state Markov setting under both complete and incomplete markets.

These examples differ in how Markov states are jumping between peace and war.

To wrap procedures for solving models, relabeling graphs so that we record government *debt* rather than government *assets*, and displaying results, we construct a Python class.

```python
class TaxSmoothingExample:
    """
    construct a tax-smoothing example, by relabeling consumption problem class.
    """
    def __init__(self, g, P, b0, states, β=.96,
                 init=0, s_path=None, N_simul=80, random_state=1):

        self.states = states # state names

        # if the path of states is not specified
        if s_path is None:
            self.cp = ConsumptionProblem(β, g, b0, P, init=init)
            self.s_path = self.cp.simulate(N_simul=N_simul, random_state=random_state)
```

(continues on next page)

(continued from previous page)

```
        # if the path of states is specified
        else:
            self.cp = ConsumptionProblem(β, g, b0, P, init=s_path[0])
            self.s_path = s_path

        # solve for complete market case
        self.T_bar, self.b = consumption_complete(self.cp)
        self.debt_value = - (β * P @ self.b).T

        # solve for incomplete market case
        self.T_path, self.asset_path, self.g_path = \
            consumption_incomplete(self.cp, self.s_path)

        # calculate returns on state-contingent debt
        self.R = ex_post_gross_return(self.b, self.cp)
        self.RT_path = cumulative_return(self.s_path, self.R)

    def display(self):

        # plot graphs
        N = len(self.T_path)

        plt.figure()
        plt.title('Tax collection paths')
        plt.plot(np.arange(N), self.T_path, label='incomplete market')
        plt.plot(np.arange(N), np.full(N, self.T_bar), label='complete market')
        plt.plot(np.arange(N), self.g_path, label='govt expenditures', alpha=.6, ls='-
↪-')
        plt.legend()
        plt.xlabel('Periods')
        plt.show()

        plt.title('Government debt paths')
        plt.plot(np.arange(N), -self.asset_path, label='incomplete market')
        plt.plot(np.arange(N), -self.b[self.s_path], label='complete market')
        plt.plot(np.arange(N), self.g_path, label='govt expenditures', ls='--')
        plt.plot(np.arange(N), self.debt_value[self.s_path], label="value of debts␣
↪today")
        plt.legend()
        plt.axhline(0, color='k', ls='--')
        plt.xlabel('Periods')
        plt.show()

        fig, ax = plt.subplots()
        ax.set_title('Cumulative return path (complete markets)')
        line1 = ax.plot(np.arange(N), self.RT_path, color='blue')[0]
        c1 = line1.get_color()
        ax.set_xlabel('Periods')
        ax.set_ylabel('Cumulative return', color=c1)

        ax_ = ax.twinx()
        line2 = ax_.plot(np.arange(N), self.g_path, ls='--', color='green')[0]
        c2 = line2.get_color()
        ax_.set_ylabel('Government expenditures', color=c2)

        plt.show()
```

(continues on next page)

(continued from previous page)

```
        # plot detailed information
        Q = self.cp.β * self.cp.P

        print(f"P \n {self.cp.P}")
        print(f"Q \n {Q}")
        print(f"Govt expenditures in {', '.join(self.states)} = {self.cp.y.flatten()}
↪")
        print(f"Constant tax collections = {self.T_bar}")
        print(f"Govt debt in {len(self.states)} states = {-self.b}")

        print("")
        print(f"Government tax collections minus debt levels in {', '.join(self.
↪states)}")
        for i in range(len(self.states)):
            TB = self.T_bar + self.b[i]
            print(f"  T+b in {self.states[i]} = {TB}")

        print("")
        print(f"Total government spending in {', '.join(self.states)}")
        for i in range(len(self.states)):
            G = self.cp.y[i, 0] + Q[i, :] @ self.b
            print(f"  {self.states[i]} = {G}")

        print("")
        print("Let's see ex-post and ex-ante returns on Arrow securities \n")

        print(f"Ex-post returns to purchase of Arrow securities:")
        for i in range(len(self.states)):
            for j in range(len(self.states)):
                if Q[i, j] != 0.:
                    print(f"  π({self.states[j]}|{self.states[i]}) = {1/Q[i, j]}")

        print("")
        exant = 1 / self.cp.β
        print(f"Ex-ante returns to purchase of Arrow securities = {exant}")

        print("")
        print("The Ex-post one-period gross return on the portfolio of government␣
↪assets")
        print(self.R)

        print("")
        print("The cumulative return earned from holding 1 unit market portfolio of␣
↪government bonds")
        print(self.RT_path[-1])
```

12.4.1 Parameters

```
γ = .1
λ = .1
ϕ = .1
θ = .1
ψ = .1
g_L = .5
g_M = .8
g_H = 1.2
β = .96
```

12.4.2 Example 1

This example is designed to produce some stylized versions of tax, debt, and deficit paths followed by the United States during and after the Civil War and also during and after World War I.

We set the Markov chain to have three states

$$P = \begin{bmatrix} 1-\lambda & \lambda & 0 \\ 0 & 1-\phi & \phi \\ 0 & 0 & 1 \end{bmatrix}$$

where the government expenditure vector $g = \begin{bmatrix} g_L & g_H & g_M \end{bmatrix}$ where $g_L < g_M < g_H$.

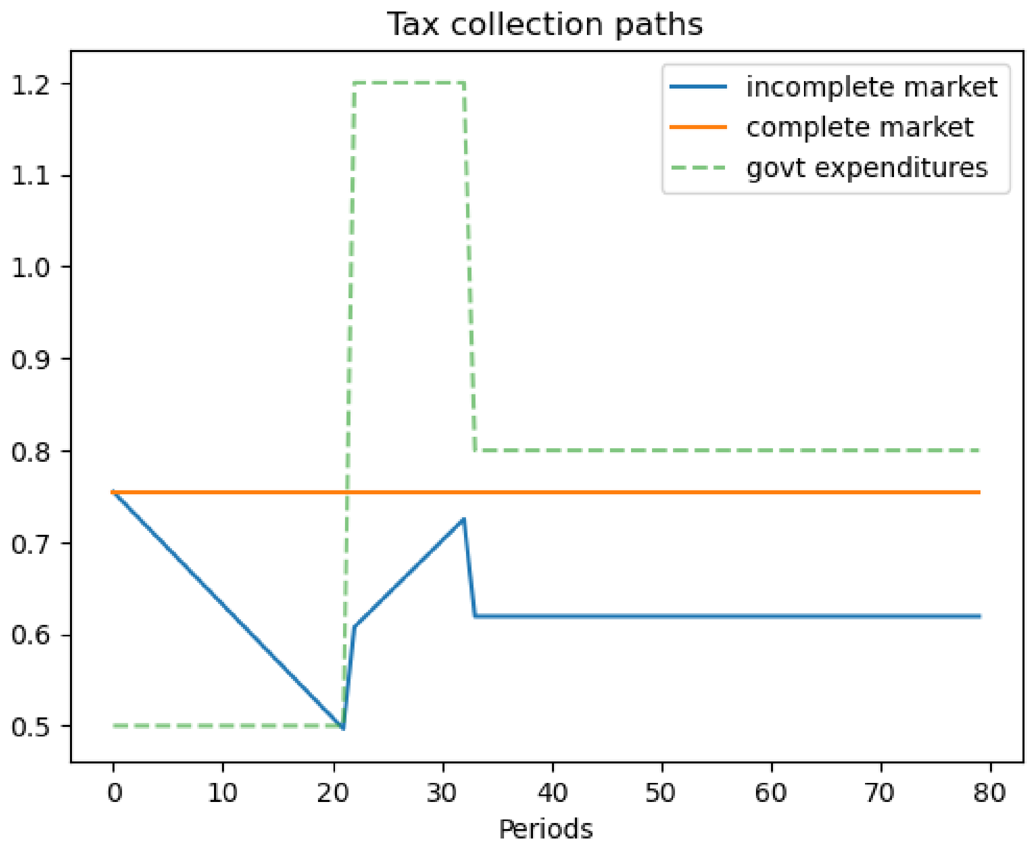
We set $b_0 = 1$ and assume that the initial Markov state is state 1 so that the system starts off in peace.

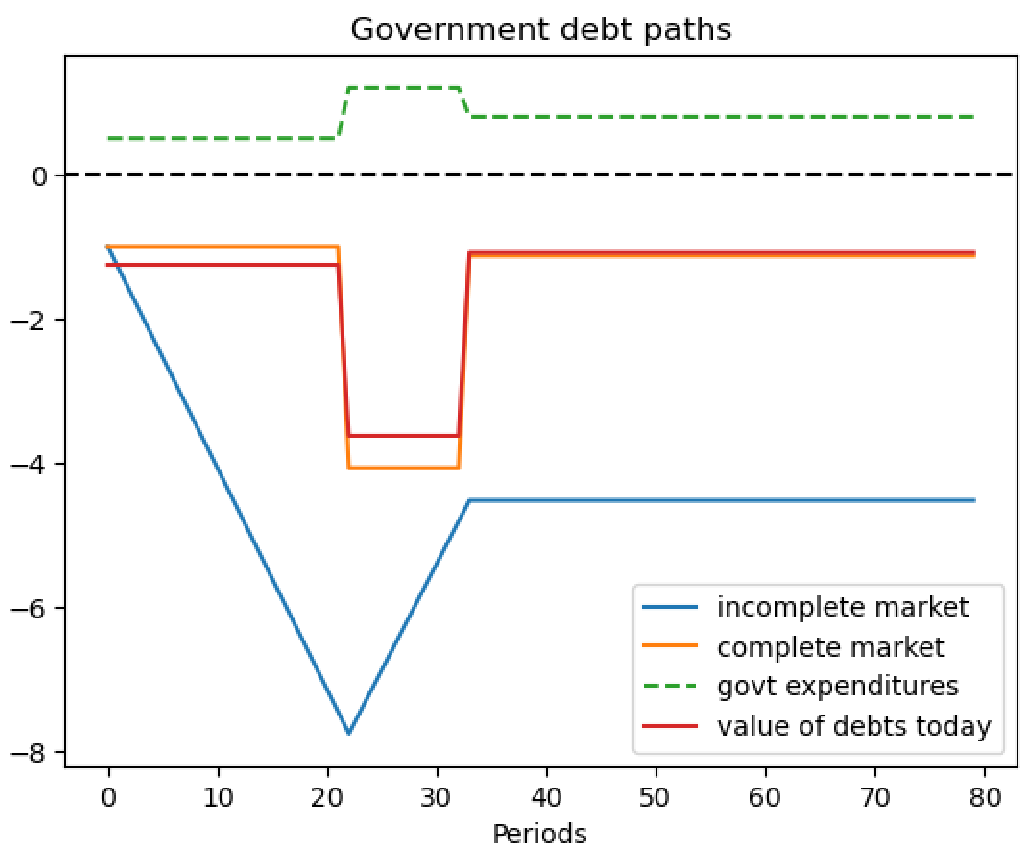
These parameters have government expenditure beginning at a low level, surging during the war, then decreasing after the war to a level that exceeds its prewar level.

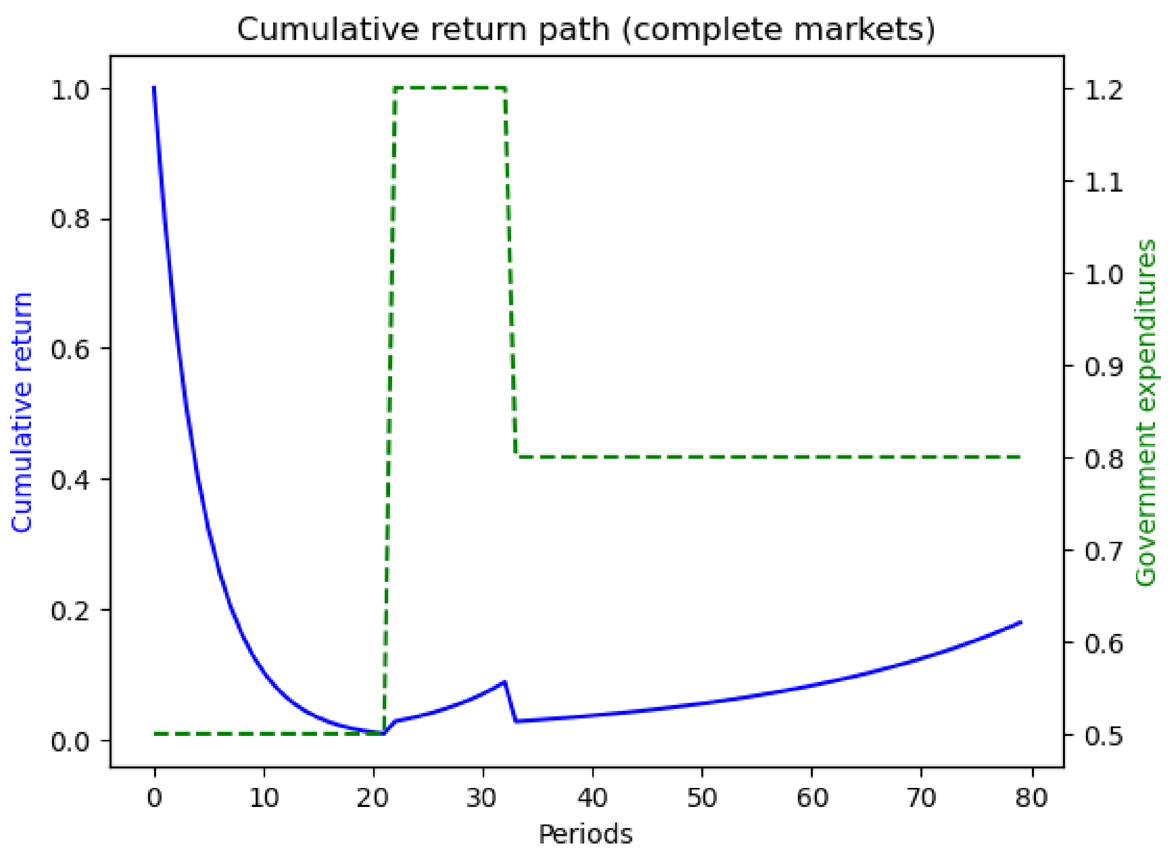
(This type of pattern occurred in the US Civil War and World War I experiences.)

```
g_ex1 = [g_L, g_H, g_M]
P_ex1 = np.array([[1-λ, λ,   0],
                  [0, 1-ϕ,   ϕ],
                  [0,    0,   1]])
b0_ex1 = 1
states_ex1 = ['peace', 'war', 'postwar']
```

```
ts_ex1 = TaxSmoothingExample(g_ex1, P_ex1, b0_ex1, states_ex1, random_state=1)
ts_ex1.display()
```


Tax collection paths
incomplete market
complete market
govt expenditures
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50
60
70
80
Periods


Government debt paths
0
−2
−4
−6
−8
0
10
20
30
40
50
60
70
80
Periods
incomplete market
complete market
govt expenditures
value of debts today



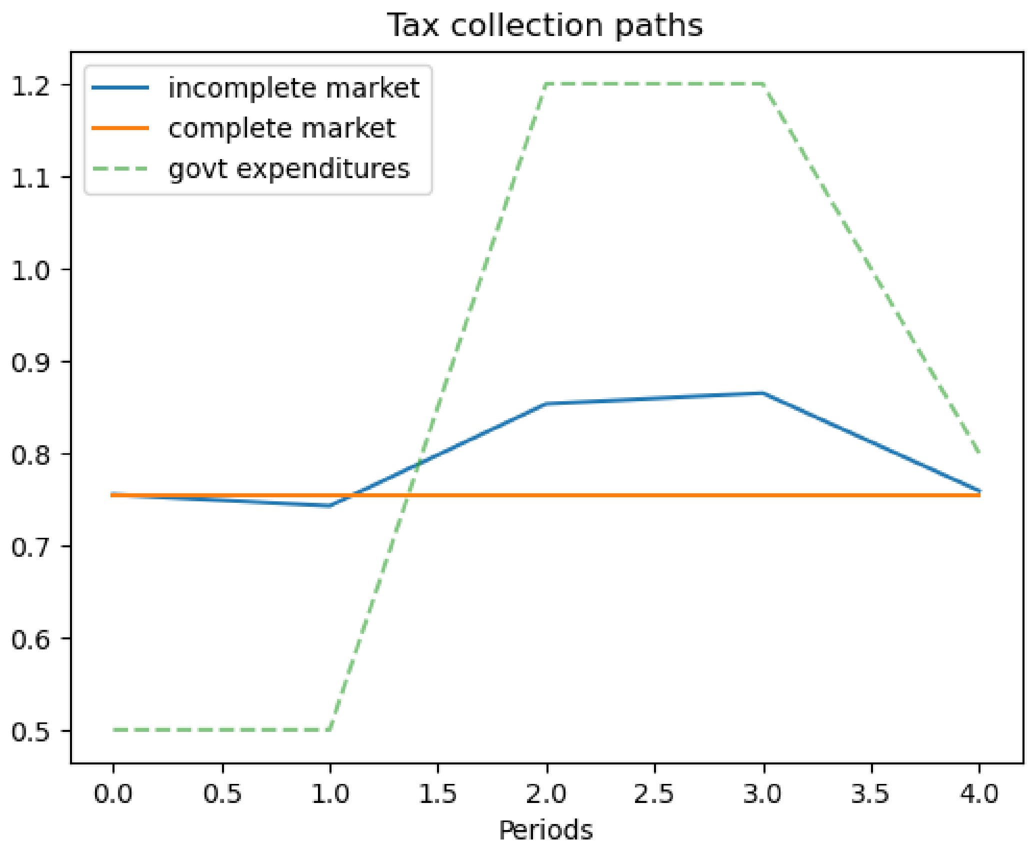
```
P
 [[0.9 0.1 0. ]
 [0.  0.9 0.1]
 [0.  0.  1. ]]
Q
 [[0.864 0.096 0.   ]
 [0.    0.864 0.096]
 [0.    0.    0.96 ]]
Govt expenditures in peace, war, postwar = [0.5 1.2 0.8]
Constant tax collections = 0.7548096885813149
Govt debt in 3 states = [-1.         -4.07093426 -1.12975779]

Government tax collections minus debt levels in peace, war, postwar
  T+b in peace = 1.754809688581315
  T+b in war = 4.825743944636679
  T+b in postwar = 1.8845674740484442

Total government spending in peace, war, postwar
  peace = 1.754809688581315
  war = 4.825743944636679
  postwar = 1.8845674740484442

Let's see ex-post and ex-ante returns on Arrow securities

Ex-post returns to purchase of Arrow securities:
  π(peace|peace) = 1.1574074074074074
  π(war|peace) = 10.416666666666666
```

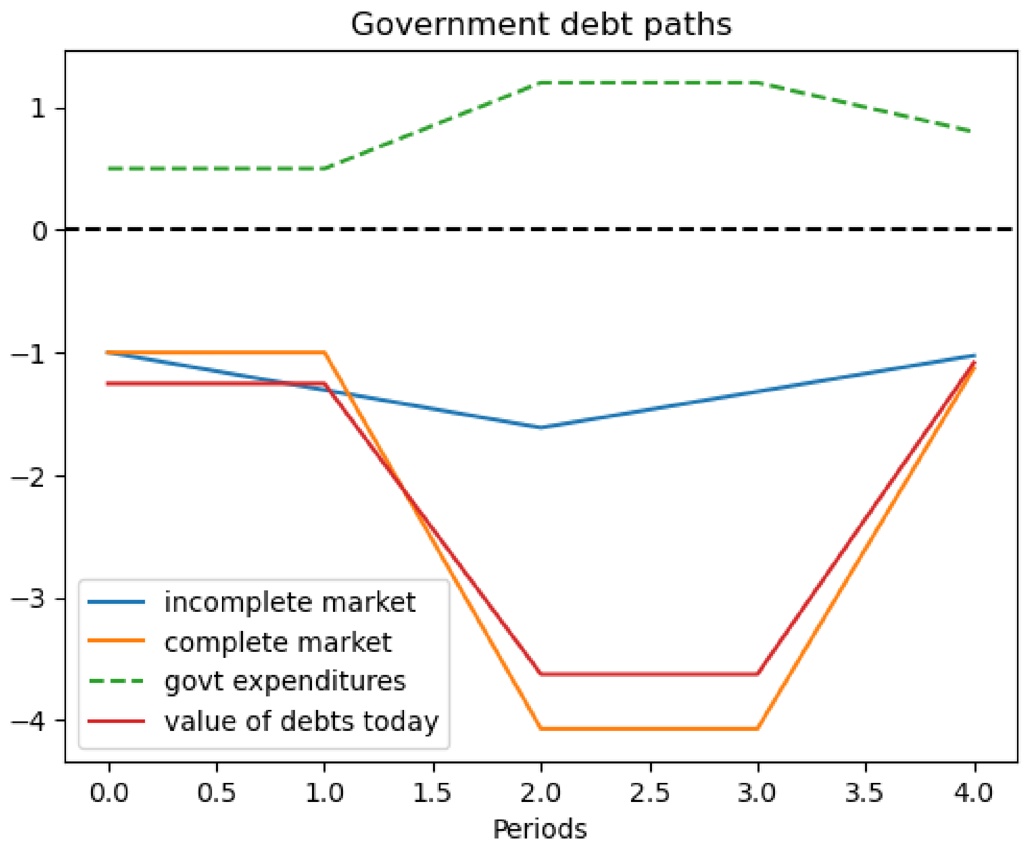
(continues on next page)

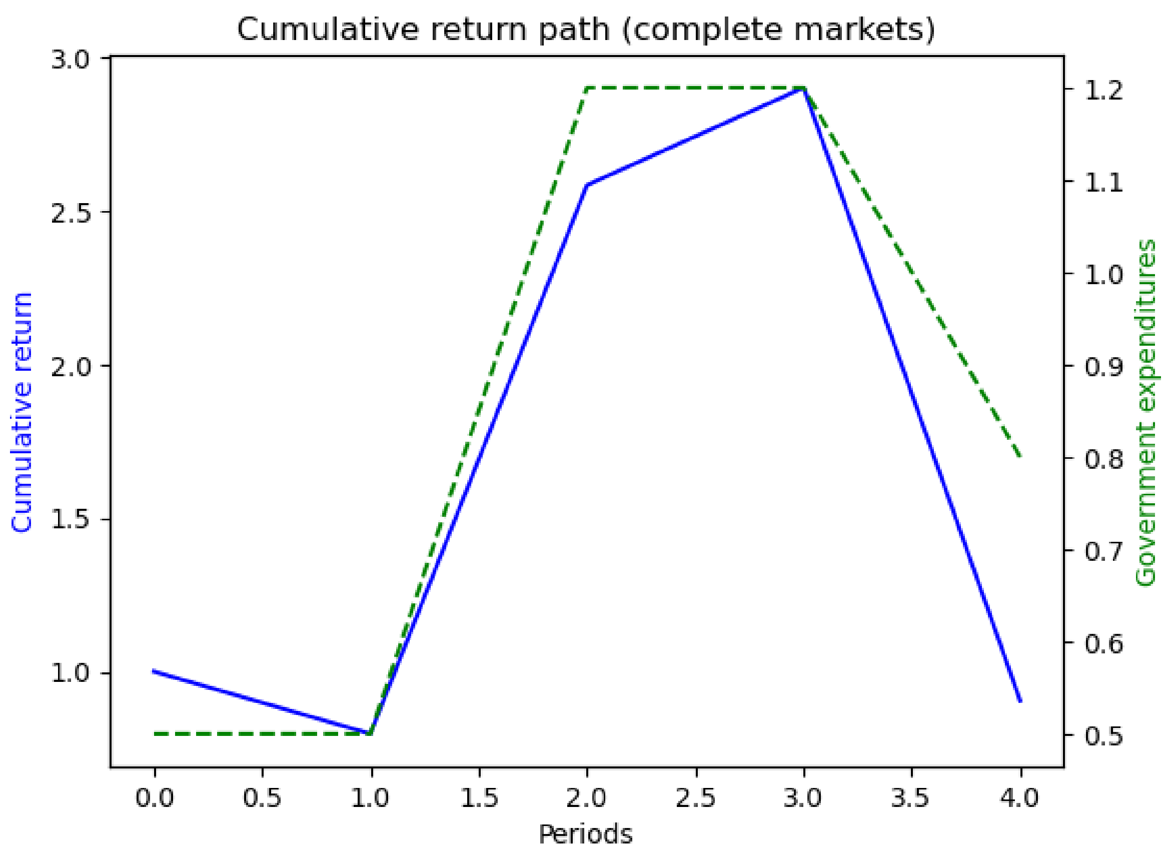
(continued from previous page)

```
  π(war|war) = 1.1574074074074074
  π(postwar|war) = 10.416666666666666
  π(postwar|postwar) = 1.0416666666666667

Ex-ante returns to purchase of Arrow securities = 1.0416666666666667

The Ex-post one-period gross return on the portfolio of government assets
[[0.7969336  3.24426428 0.        ]
 [0.         1.12278592 0.31159337]
 [0.         0.         1.04166667]]

The cumulative return earned from holding 1 unit market portfolio of government␣
↪bonds
0.17908622141460231
```

```
# The following shows the use of the wrapper class when a specific state path is given
s_path = [0, 0, 1, 1, 2]
ts_s_path = TaxSmoothingExample(g_ex1, P_ex1, b0_ex1, states_ex1, s_path=s_path)
ts_s_path.display()
```




Government debt paths
incomplete market
complete market
govt expenditures
value of debts today
Periods
1
0
−1
−2
−3
−4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



```
P
 [[0.9 0.1 0. ]
 [0.  0.9 0.1]
 [0.  0.  1. ]]
Q
 [[0.864 0.096 0.   ]
 [0.    0.864 0.096]
 [0.    0.    0.96 ]]
Govt expenditures in peace, war, postwar = [0.5 1.2 0.8]
Constant tax collections = 0.7548096885813149
Govt debt in 3 states = [-1.         -4.07093426 -1.12975779]

Government tax collections minus debt levels in peace, war, postwar
  T+b in peace = 1.754809688581315
  T+b in war = 4.825743944636679
  T+b in postwar = 1.8845674740484442

Total government spending in peace, war, postwar
  peace = 1.754809688581315
  war = 4.825743944636679
  postwar = 1.8845674740484442

Let's see ex-post and ex-ante returns on Arrow securities

Ex-post returns to purchase of Arrow securities:
  π(peace|peace) = 1.1574074074074074
  π(war|peace) = 10.416666666666666
```

(continues on next page)

(continued from previous page)

```
  π(war|war) = 1.1574074074074074
  π(postwar|war) = 10.416666666666666
  π(postwar|postwar) = 1.0416666666666667

Ex-ante returns to purchase of Arrow securities = 1.0416666666666667

The Ex-post one-period gross return on the portfolio of government assets
[[0.7969336  3.24426428 0.        ]
 [0.         1.12278592 0.31159337]
 [0.         0.         1.04166667]]

The cumulative return earned from holding 1 unit market portfolio of government␣
 ↪bonds
0.9045311615620277
```

12.4.3 Example 2

This example captures a peace followed by a war, eventually followed by a permanent peace .

Here we set

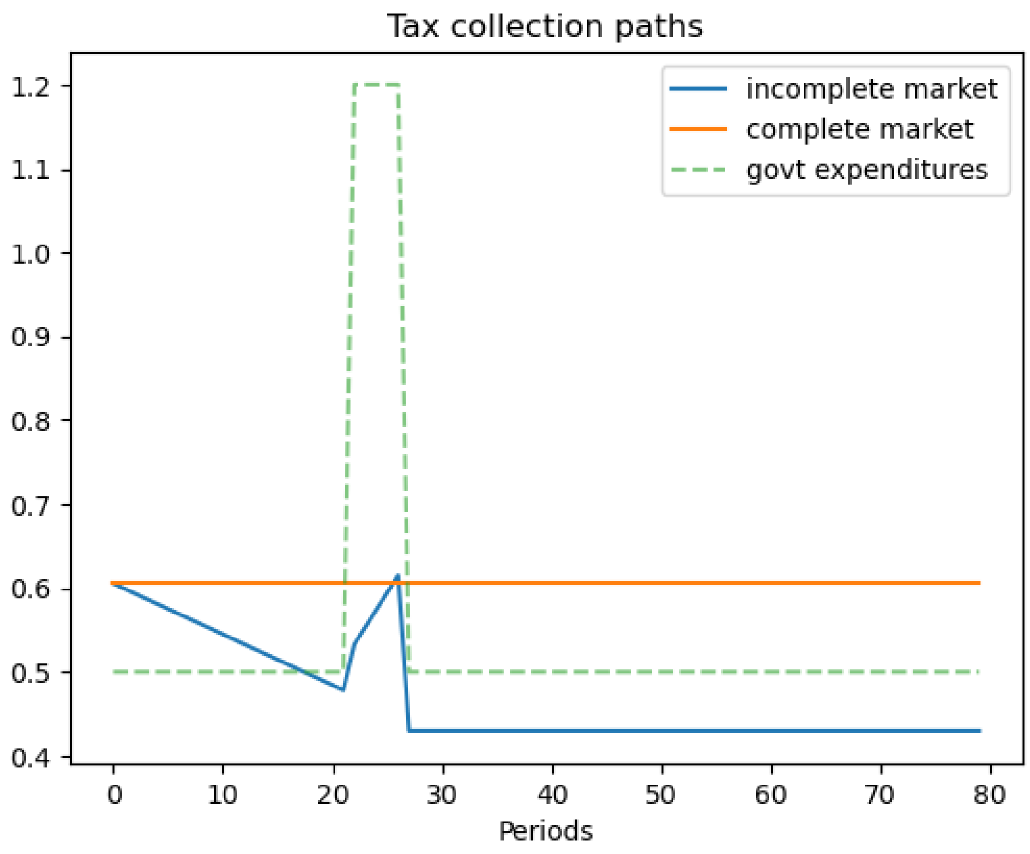
$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-\gamma & \gamma \\ \phi & 0 & 1-\phi \end{bmatrix}$$

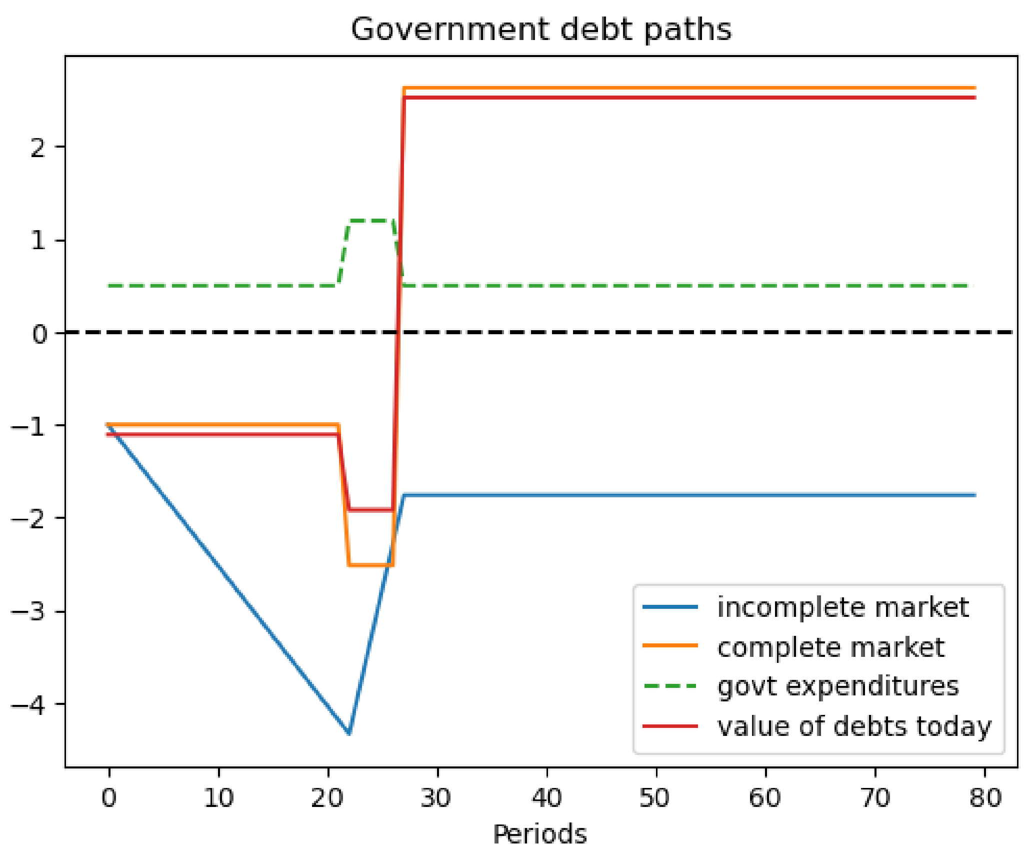
where the government expenditure vector $g = \begin{bmatrix} g_L & g_L & g_H \end{bmatrix}$ and where $g_L < g_H$.

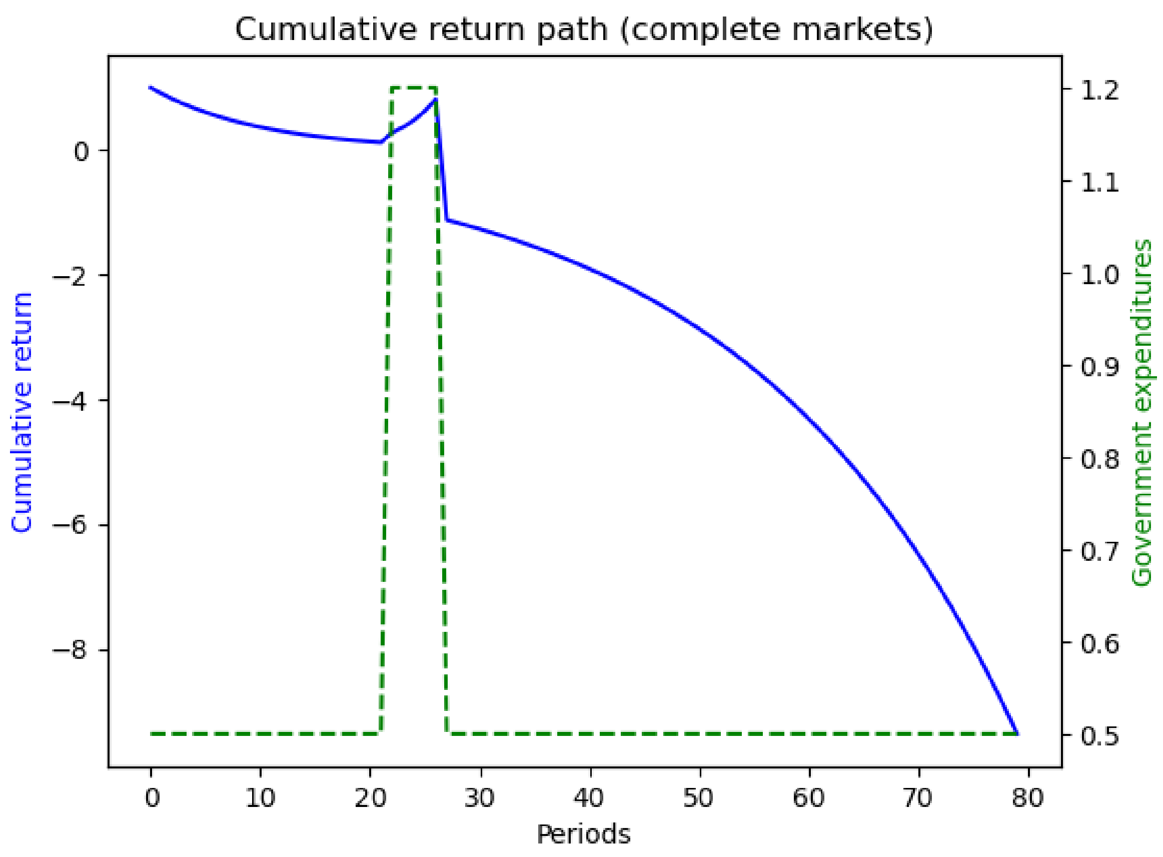
We assume $b_0 = 1$ and that the initial Markov state is state 2 so that the system starts off in a temporary peace.

```
g_ex2 = [g_L, g_L, g_H]
P_ex2 = np.array([[1,    0,     0],
                  [0, 1-γ,     γ],
                  [ϕ,    0, 1-ϕ]])
b0_ex2 = 1
states_ex2 = ['peace', 'temporary peace', 'war']
```

```
ts_ex2 = TaxSmoothingExample(g_ex2, P_ex2, b0_ex2, states_ex2, init=1, random_state=1)
ts_ex2.display()
```


Tax collection paths
incomplete market
complete market
govt expenditures
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50
60
70
80
Periods


Government debt paths
2
1
0
−1
−2
−3
−4
0
10
20
30
40
50
60
70
80
Periods
incomplete market
complete market
govt expenditures
value of debts today



```
P
 [[1.  0.  0. ]
 [0.  0.9 0.1]
 [0.1 0.  0.9]]
Q
 [[0.96  0.    0.   ]
 [0.    0.864 0.096]
 [0.096 0.    0.864]]
Govt expenditures in peace, temporary peace, war = [0.5 0.5 1.2]
Constant tax collections = 0.6053287197231834
Govt debt in 3 states = [ 2.63321799 -1.         -2.51384083]

Government tax collections minus debt levels in peace, temporary peace, war
  T+b in peace = -2.0278892733564
  T+b in temporary peace = 1.6053287197231834
  T+b in war = 3.1191695501730106

Total government spending in peace, temporary peace, war
  peace = -2.0278892733564
  temporary peace = 1.6053287197231834
  war = 3.1191695501730106

Let's see ex-post and ex-ante returns on Arrow securities

Ex-post returns to purchase of Arrow securities:
  π(peace|peace) = 1.0416666666666667
  π(temporary peace|temporary peace) = 1.1574074074074074
```

(continues on next page)

(continued from previous page)

```
  π(war|temporary peace) = 10.416666666666666
  π(peace|war) = 10.416666666666666
  π(war|war) = 1.1574074074074074

Ex-ante returns to purchase of Arrow securities = 1.0416666666666667

The Ex-post one-period gross return on the portfolio of government assets
[[ 1.04166667  0.          0.        ]
 [ 0.          0.90470824  2.27429251]
 [-1.37206116  0.          1.30985865]]

The cumulative return earned from holding 1 unit market portfolio of government␣
 ↪bonds
-9.368991732594216
```

12.4.4 Example 3

This example features a situation in which one of the states is a war state with no hope of peace next period, while another state is a war state with a positive probability of peace next period.

The Markov chain is:

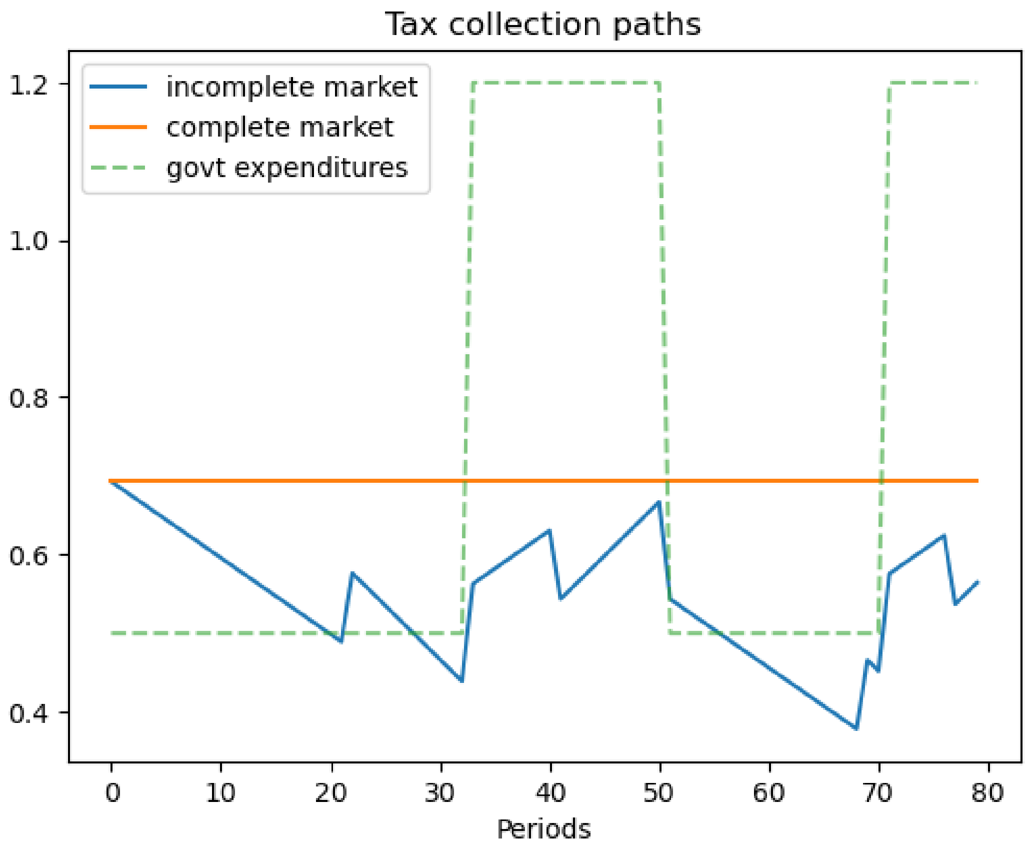
$$P = \begin{bmatrix} 1-\lambda & \lambda & 0 & 0 \\ 0 & 1-\phi & \phi & 0 \\ 0 & 0 & 1-\psi & \psi \\ \theta & 0 & 0 & 1-\theta \end{bmatrix}$$

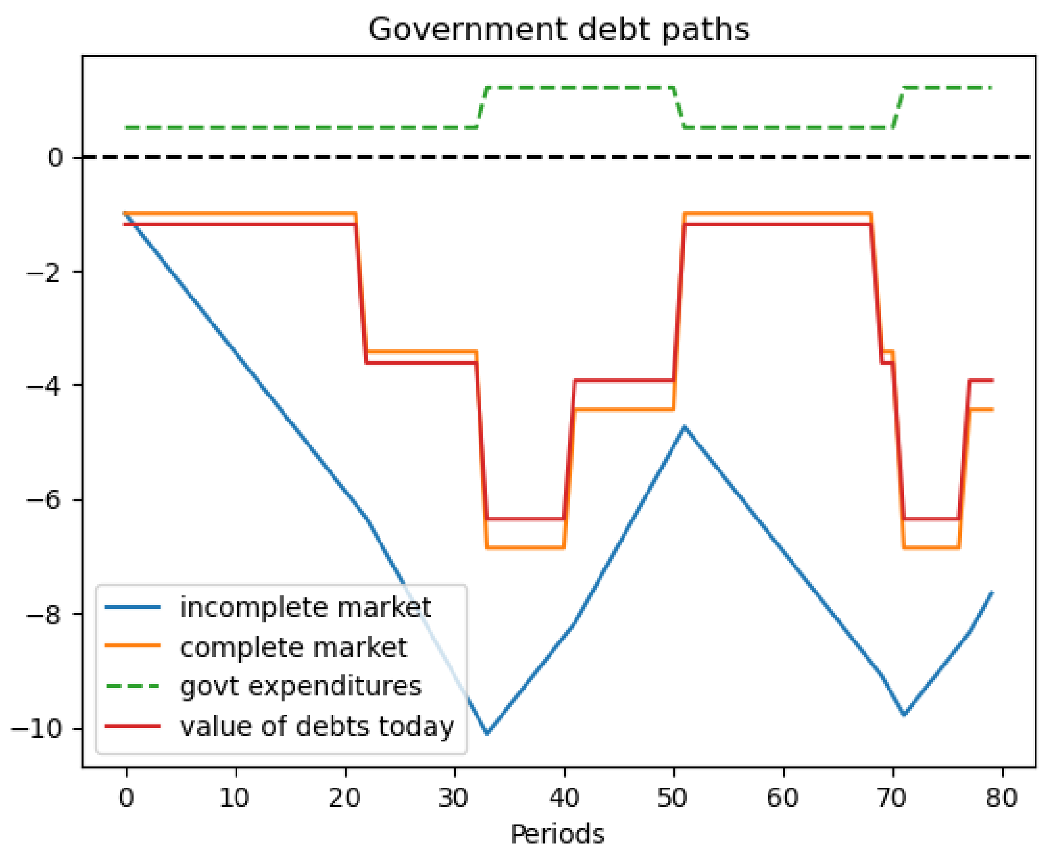
with government expenditure levels for the four states being $[g_L \quad g_L \quad g_H \quad g_H]$ where $g_L < g_H$.

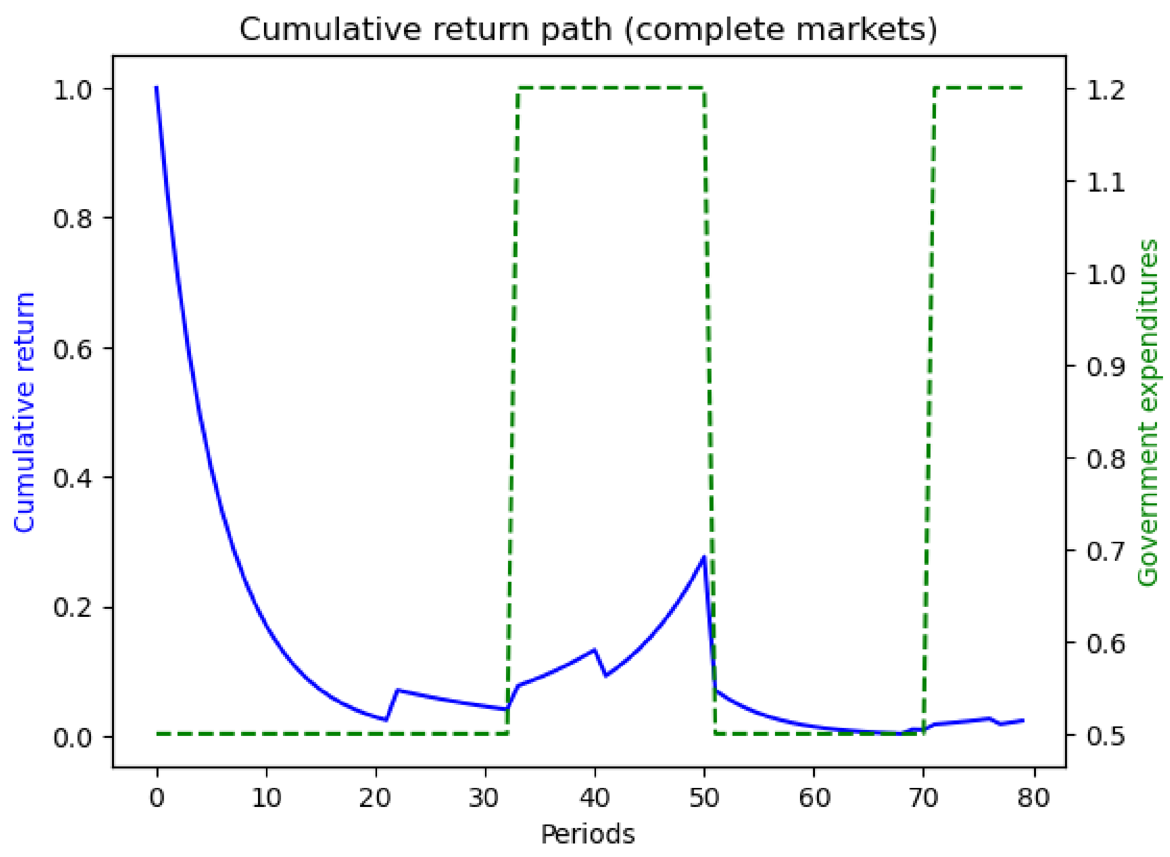
We start with $b_0 = 1$ and $s_0 = 1$.

```
g_ex3 = [g_L, g_L, g_H, g_H]
P_ex3 = np.array([[1-λ,  λ,    0,     0],
                  [0,  1-ϕ,    ϕ,     0],
                  [0,    0,  1-ψ,     ψ],
                  [θ,    0,    0,  1-θ ]])
b0_ex3 = 1
states_ex3 = ['peace1', 'peace2', 'war1', 'war2']
```

```
ts_ex3 = TaxSmoothingExample(g_ex3, P_ex3, b0_ex3, states_ex3, random_state=1)
ts_ex3.display()
```


Tax collection paths
incomplete market
complete market
govt expenditures
1.2
1.0
0.8
0.6
0.4
0
10
20
30
40
50
60
70
80
Periods


Government debt paths
0
−2
−4
−6
−8
−10
0
10
20
30
40
50
60
70
80
Periods
incomplete market
complete market
govt expenditures
value of debts today



```
P
 [[0.9 0.1 0.  0. ]
 [0.  0.9 0.1 0. ]
 [0.  0.  0.9 0.1]
 [0.1 0.  0.  0.9]]
Q
 [[0.864 0.096 0.    0.   ]
 [0.    0.864 0.096 0.   ]
 [0.    0.    0.864 0.096]
 [0.096 0.    0.    0.864]]
Govt expenditures in peace1, peace2, war1, war2 = [0.5 0.5 1.2 1.2]
Constant tax collections = 0.6927944572748268
Govt debt in 4 states = [-1.         -3.42494226 -6.86027714 -4.43533487]

Government tax collections minus debt levels in peace1, peace2, war1, war2
  T+b in peace1 = 1.6927944572748268
  T+b in peace2 = 4.117736720554273
  T+b in war1 = 7.553071593533488
  T+b in war2 = 5.128129330254041

Total government spending in peace1, peace2, war1, war2
  peace1 = 1.6927944572748268
  peace2 = 4.117736720554273
  war1 = 7.553071593533487
  war2 = 5.128129330254041

Let's see ex-post and ex-ante returns on Arrow securities
```

(continues on next page)

(continued from previous page)

```
Ex-post returns to purchase of Arrow securities:
  π(peace1|peace1) = 1.1574074074074074
  π(peace2|peace1) = 10.416666666666666
  π(peace2|peace2) = 1.1574074074074074
  π(war1|peace2) = 10.416666666666666
  π(war1|war1) = 1.1574074074074074
  π(war2|war1) = 10.416666666666666
  π(peace1|war2) = 10.416666666666666
  π(war2|war2) = 1.1574074074074074

Ex-ante returns to purchase of Arrow securities = 1.0416666666666667

The Ex-post one-period gross return on the portfolio of government assets
[[0.83836741 2.87135998 0.         0.        ]
 [0.         0.94670854 1.89628977 0.        ]
 [0.         0.         1.07983627 0.69814023]
 [0.2545741  0.         0.         1.1291214 ]]

The cumulative return earned from holding 1 unit market portfolio of government␣
↪bonds
0.02371440178864222
```

12.4.5 Example 4

Here the Markov chain is:

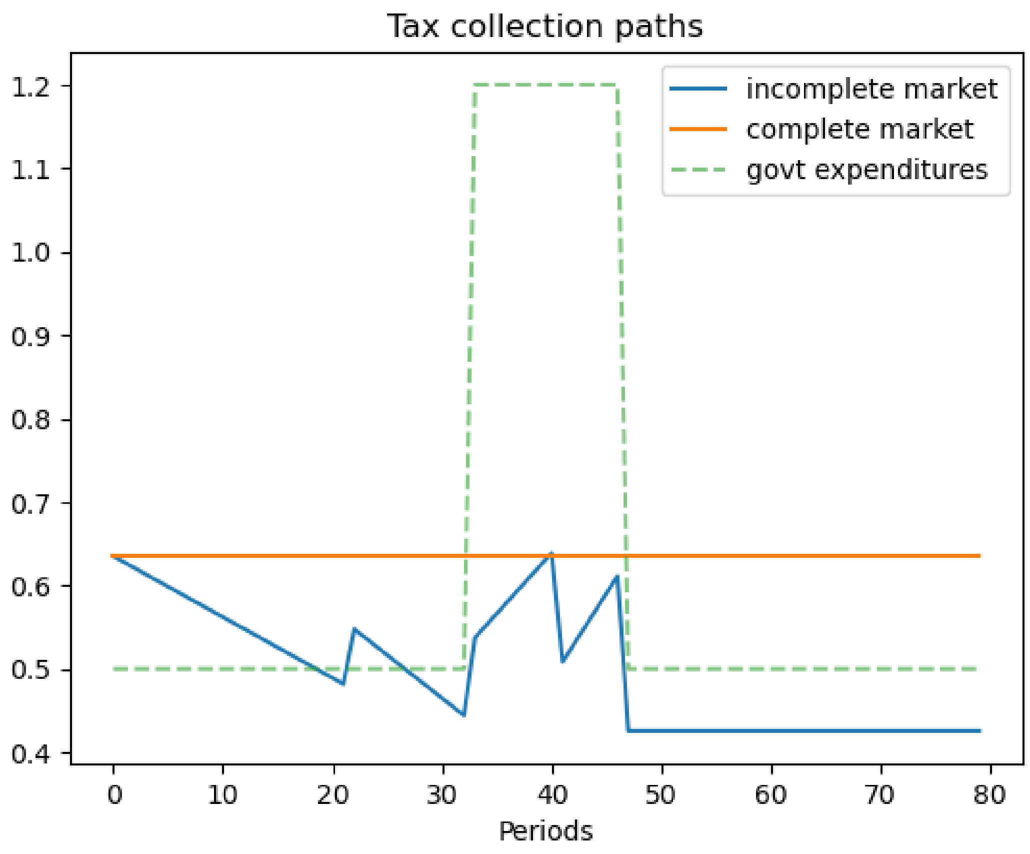
$$P = \begin{bmatrix} 1-\lambda & \lambda & 0 & 0 & 0 \\ 0 & 1-\phi & \phi & 0 & 0 \\ 0 & 0 & 1-\psi & \psi & 0 \\ 0 & 0 & 0 & 1-\theta & \theta \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

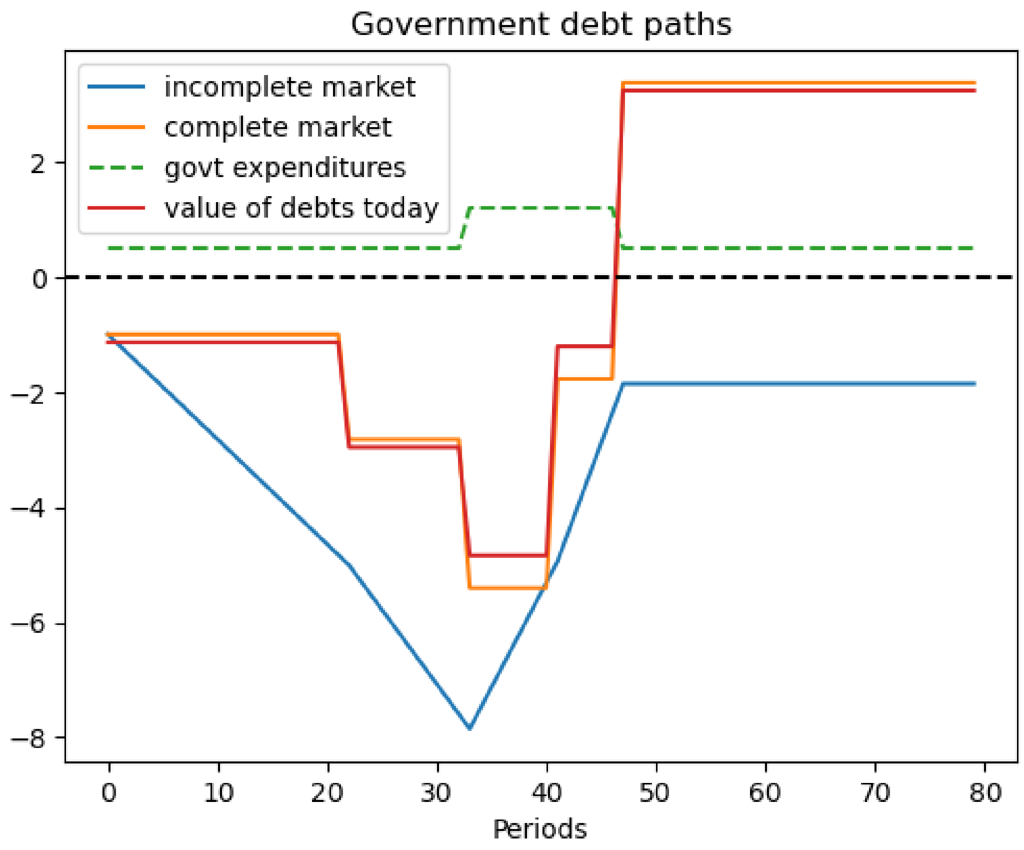
with government expenditure levels for the five states being $\begin{bmatrix} g_L & g_L & g_H & g_H & g_L \end{bmatrix}$ where $g_L < g_H$.

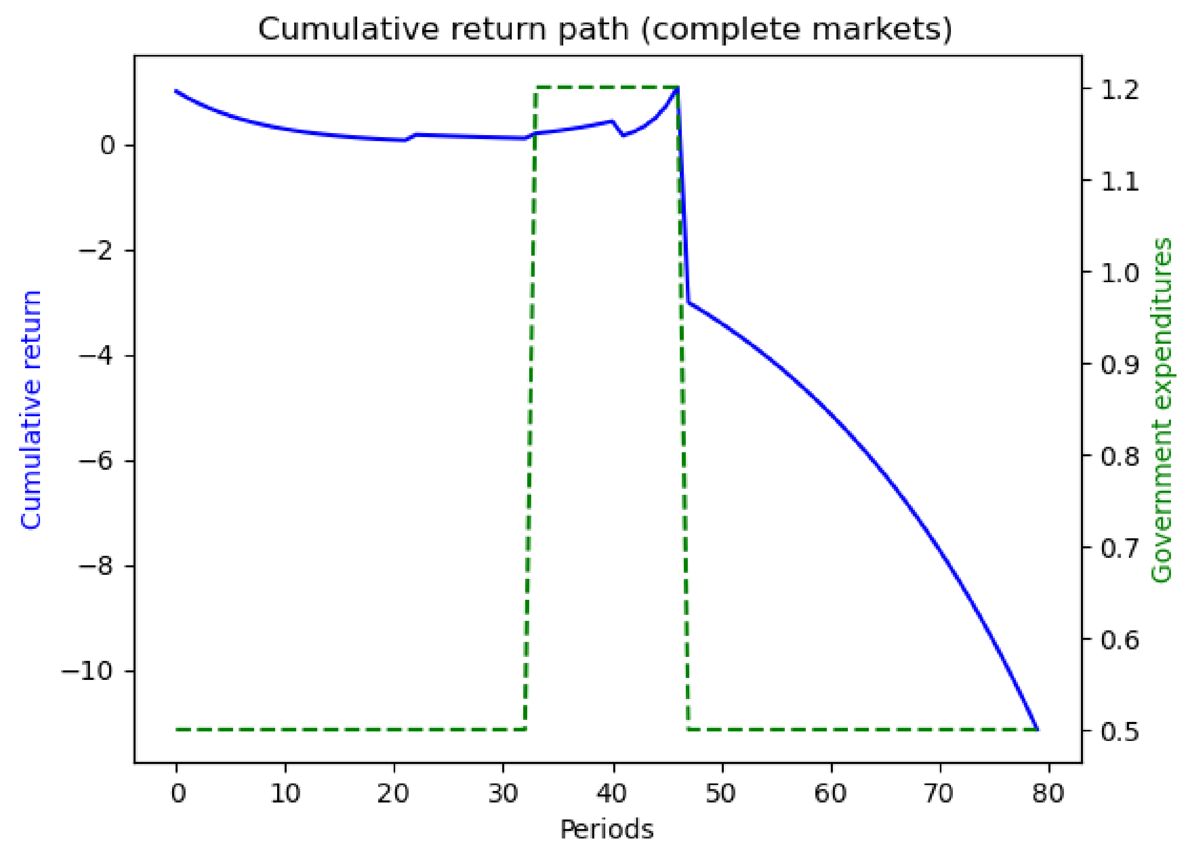
We ssume that $b_0 = 1$ and $s_0 = 1$.

```
g_ex4 = [g_L, g_L, g_H, g_H, g_L]
P_ex4 = np.array([[1-λ,  λ,    0,      0,     0],
                  [0,  1-ϕ,    ϕ,      0,     0],
                  [0,    0,  1-ψ,      ψ,     0],
                  [0,    0,    0,    1-θ,     θ],
                  [0,    0,    0,      0,     1]])
b0_ex4 = 1
states_ex4 = ['peace1', 'peace2', 'war1', 'war2', 'permanent peace']
```

```
ts_ex4 = TaxSmoothingExample(g_ex4, P_ex4, b0_ex4, states_ex4, random_state=1)
ts_ex4.display()
```


Tax collection paths
incomplete market
complete market
govt expenditures
Periods
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50
60
70
80


Government debt paths
incomplete market
complete market
govt expenditures
value of debts today
2
0
−2
−4
−6
−8
0
10
20
30
40
50
60
70
80
Periods



```
P
 [[0.9 0.1 0.  0.  0. ]
 [0.  0.9 0.1 0.  0. ]
 [0.  0.  0.9 0.1 0. ]
 [0.  0.  0.  0.9 0.1]
 [0.  0.  0.  0.  1. ]]
Q
 [[0.864 0.096 0.    0.    0.   ]
 [0.    0.864 0.096 0.    0.   ]
 [0.    0.    0.864 0.096 0.   ]
 [0.    0.    0.    0.864 0.096]
 [0.    0.    0.    0.    0.96 ]]
Govt expenditures in peace1, peace2, war1, war2, permanent peace = [0.5 0.5 1.2 1.
↪2 0.5]
Constant tax collections = 0.6349979047185738
Govt debt in 5 states = [-1.         -2.82289484 -5.4053292  -1.77211121  3.
↪37494762]

Government tax collections minus debt levels in peace1, peace2, war1, war2,␣
↪permanent peace
  T+b in peace1 = 1.6349979047185736
  T+b in peace2 = 3.4578927455370505
  T+b in war1 = 6.040327103363229
  T+b in war2 = 2.4071091102836433
  T+b in permanent peace = -2.7399497132457697

Total government spending in peace1, peace2, war1, war2, permanent peace
```

(continues on next page)

(continued from previous page)

```
  peace1 = 1.6349979047185736
  peace2 = 3.457892745537051
  war1 = 6.040327103363228
  war2 = 2.407109110283643
  permanent peace = -2.7399497132457697

Let's see ex-post and ex-ante returns on Arrow securities

Ex-post returns to purchase of Arrow securities:
  π(peace1|peace1) = 1.1574074074074074
  π(peace2|peace1) = 10.416666666666666
  π(peace2|peace2) = 1.1574074074074074
  π(war1|peace2) = 10.416666666666666
  π(war1|war1) = 1.1574074074074074
  π(war2|war1) = 10.416666666666666
  π(war2|war2) = 1.1574074074074074
  π(permanent peace|war2) = 10.416666666666666
  π(permanent peace|permanent peace) = 1.0416666666666667

Ex-ante returns to purchase of Arrow securities = 1.0416666666666667

The Ex-post one-period gross return on the portfolio of government assets
[[ 0.8810589   2.48713661  0.          0.          0.        ]
 [ 0.          0.95436011  1.82742569  0.          0.        ]
 [ 0.          0.          1.11672808  0.36611394  0.        ]
 [ 0.          0.          0.          1.46806216 -2.79589276]
 [ 0.          0.          0.          0.          1.04166667]]

The cumulative return earned from holding 1 unit market portfolio of government␣
↪bonds
-11.132109773063616
```

12.4.6 Example 5

The example captures a case when the system follows a deterministic path from peace to war, and back to peace again.

Since there is no randomness, the outcomes in complete markets setting should be the same as in incomplete markets setting.

The Markov chain is:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

with government expenditure levels for the seven states being $[g_L \quad g_L \quad g_H \quad g_H \quad g_H \quad g_H \quad g_L]$ where $g_L < g_H$. Assume $b_0 = 1$ and $s_0 = 1$.

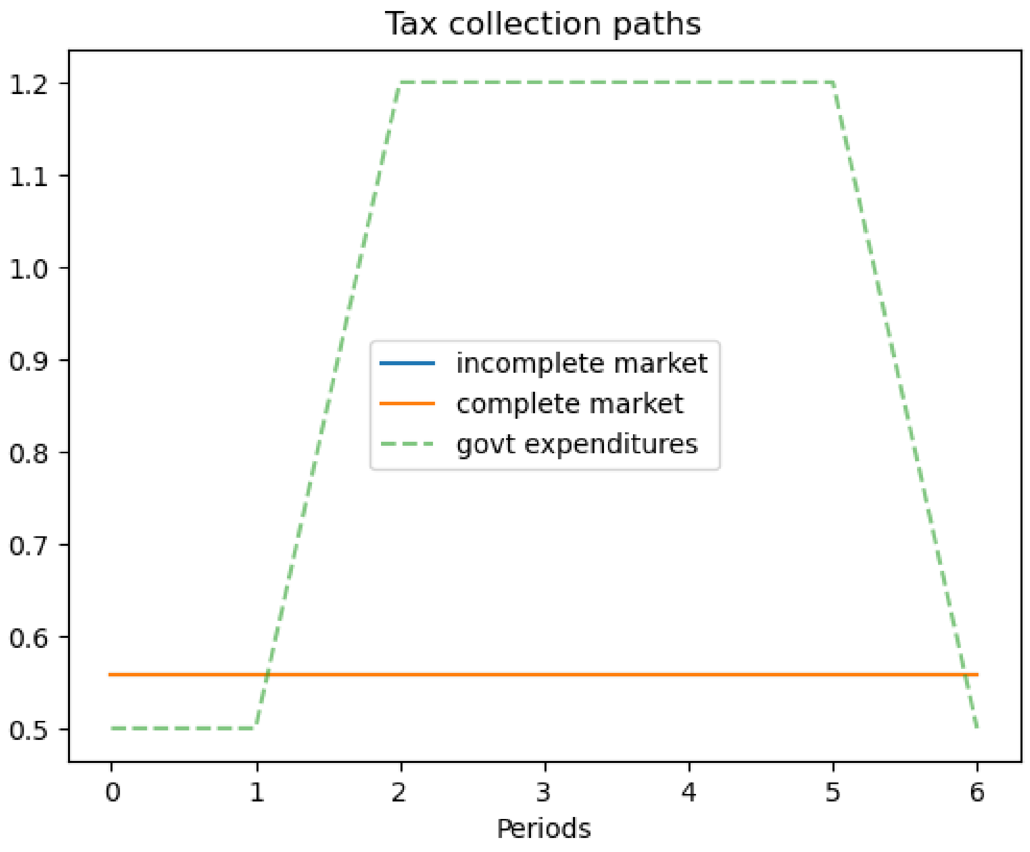
```
g_ex5 = [g_L, g_L, g_H, g_H, g_H, g_H, g_L]
P_ex5 = np.array([[0, 1, 0, 0, 0, 0, 0],
```

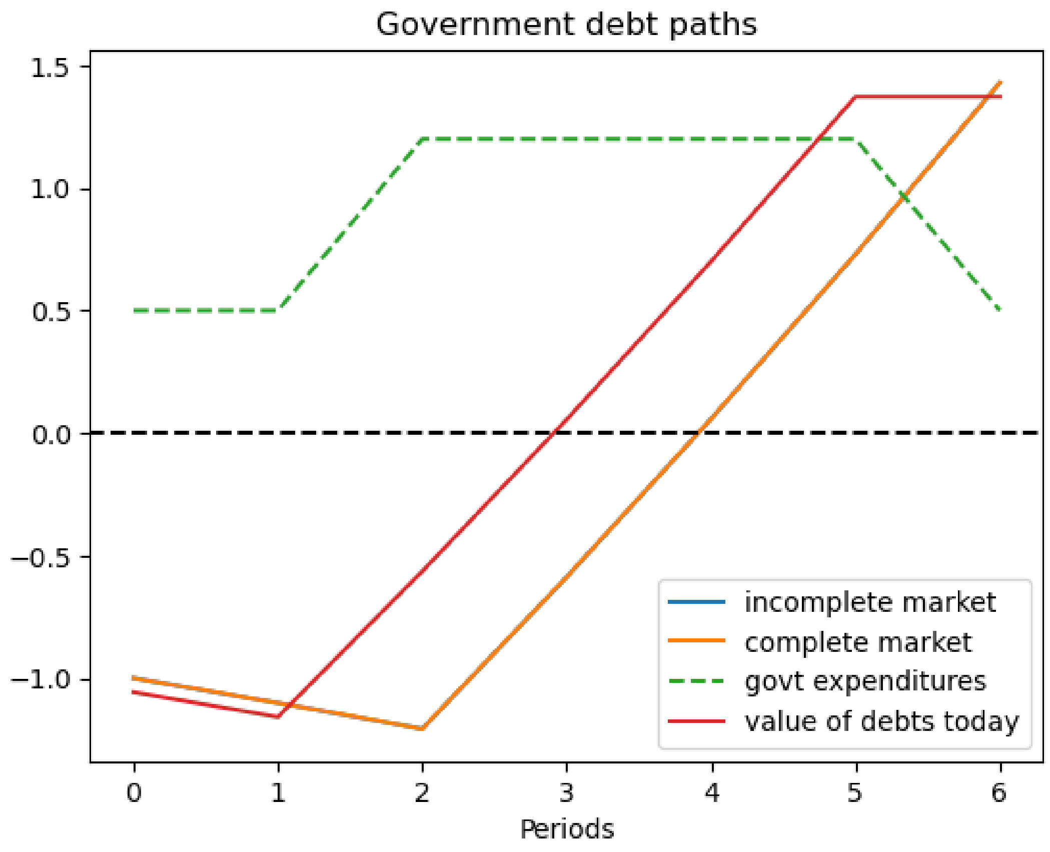
(continues on next page)

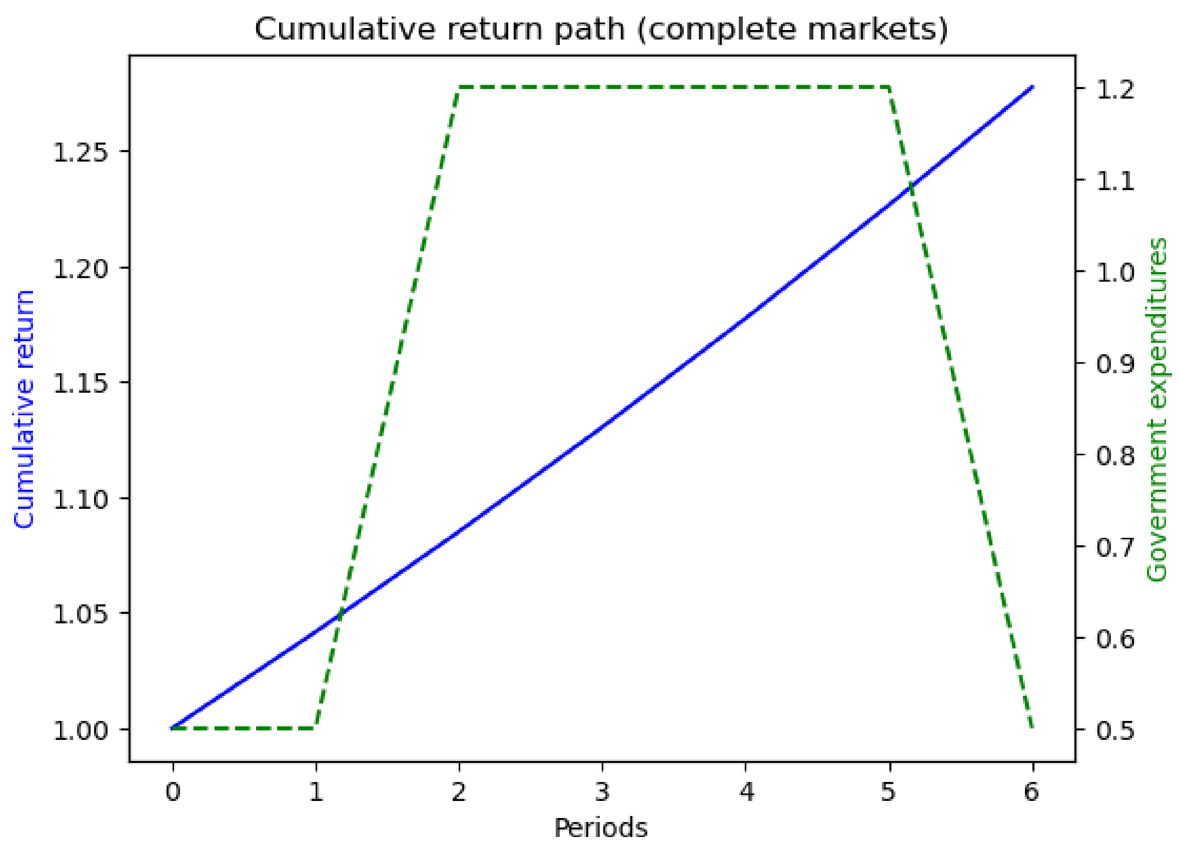
(continued from previous page)

```
                    [0, 0, 1, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0],
                    [0, 0, 0, 0, 1, 0, 0],
                    [0, 0, 0, 0, 0, 1, 0],
                    [0, 0, 0, 0, 0, 0, 1],
                    [0, 0, 0, 0, 0, 0, 1]])
b0_ex5 = 1
states_ex5 = ['peace1', 'peace2', 'war1', 'war2', 'war3', 'permanent peace']
```

```
ts_ex5 = TaxSmoothingExample(g_ex5, P_ex5, b0_ex5, states_ex5, N_simul=7, random_
 ↪state=1)
ts_ex5.display()
```




Government debt paths
1.5
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6
Periods
incomplete market
complete market
govt expenditures
value of debts today



```
P
 [[0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0]
 [0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0]
 [0 0 0 0 0 0 1]
 [0 0 0 0 0 0 1]]
Q
 [[0.   0.96 0.   0.   0.   0.   0.  ]
 [0.   0.   0.96 0.   0.   0.   0.  ]
 [0.   0.   0.   0.96 0.   0.   0.  ]
 [0.   0.   0.   0.   0.96 0.   0.  ]
 [0.   0.   0.   0.   0.   0.96 0.  ]
 [0.   0.   0.   0.   0.   0.   0.96]
 [0.   0.   0.   0.   0.   0.   0.96]]
Govt expenditures in peace1, peace2, war1, war2, war3, permanent peace = [0.5 0.5␣
 ↪1.2 1.2 1.2 1.2 0.5]
Constant tax collections = 0.5571895472128001
Govt debt in 6 states = [-1.         -1.10123911 -1.20669652 -0.58738132  0.
 ↪05773868  0.72973868
  1.42973868]

Government tax collections minus debt levels in peace1, peace2, war1, war2, war3,␣
 ↪permanent peace
  T+b in peace1 = 1.5571895472128001
  T+b in peace2 = 1.6584286588928001
```

(continues on next page)

(continued from previous page)

```
  T+b in war1 = 1.7638860668928005
  T+b in war2 = 1.1445708668928003
  T+b in war3 = 0.4994508668928004
  T+b in permanent peace = -0.17254913310719955

Total government spending in peace1, peace2, war1, war2, war3, permanent peace
  peace1 = 1.5571895472128
  peace2 = 1.6584286588928003
  war1 = 1.7638860668928
  war2 = 1.1445708668928003
  war3 = 0.49945086689280027
  permanent peace = -0.17254913310719933

Let's see ex-post and ex-ante returns on Arrow securities

Ex-post returns to purchase of Arrow securities:
  π(peace2|peace1) = 1.0416666666666667
  π(war1|peace2) = 1.0416666666666667
  π(war2|war1) = 1.0416666666666667
  π(war3|war2) = 1.0416666666666667
  π(permanent peace|war3) = 1.0416666666666667

Ex-ante returns to purchase of Arrow securities = 1.0416666666666667

The Ex-post one-period gross return on the portfolio of government assets
[[0.         1.04166667 0.         0.         0.         0.
  0.        ]
 [0.         0.         1.04166667 0.         0.         0.
  0.        ]
 [0.         0.         0.         1.04166667 0.         0.
  0.        ]
 [0.         0.         0.         0.         1.04166667 0.
  0.        ]
 [0.         0.         0.         0.         0.         1.04166667
  0.        ]
 [0.         0.         0.         0.         0.         0.
  1.04166667]
 [0.         0.         0.         0.         0.         0.
  1.04166667]]

The cumulative return earned from holding 1 unit market portfolio of government␣
 ↪bonds
1.2775343959060068
```

12.4.7 Continuous-State Gaussian Model

To construct a tax-smoothing version of the complete markets consumption-smoothing model with a continuous state space that we presented in the lecture *consumption smoothing with complete and incomplete markets*, we simply relabel variables.

Thus, a government faces a sequence of budget constraints

$$T_t + b_t = g_t + \beta \mathbb{E}_t b_{t+1}, \quad t \geq 0$$

where T_t is tax revenues, b_t are receipts at t from contingent claims that the government had *purchased* at time $t-1$, and

$$\beta \mathbb{E}_t b_{t+1} \equiv \int q_{t+1}(x_{t+1}|x_t) b_{t+1}(x_{t+1}) dx_{t+1}$$

is the value of time $t+1$ state-contingent claims purchased by the government at time t.

As above with the consumption-smoothing model, we can solve the time t budget constraint forward to obtain

$$b_t = \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j (g_{t+j} - T_{t+j})$$

which can be rearranged to become

$$\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j g_{t+j} = b_t + \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j T_{t+j}$$

which states that the present value of government purchases equals the value of government assets at t plus the present value of tax receipts.

With these relabelings, examples presented in *consumption smoothing with complete and incomplete markets* can be interpreted as tax-smoothing models.

Returns: In the continuous state version of our incomplete markets model, the ex post one-period gross rate of return on the government portfolio equals

$$R(x_{t+1}|x_t) = \frac{b(x_{t+1})}{\beta E b(x_{t+1})|x_t}$$

Related Lectures

Throughout this lecture, we have taken one-period interest rates and Arrow security prices as exogenous objects determined outside the model and specified them in ways designed to align our models closely with the consumption smoothing model of Barro [Barro, 1979].

Other lectures make these objects endogenous and describe how a government optimally manipulates prices of government debt, albeit indirectly via effects distorting taxes have on equilibrium prices and allocations.

In *optimal taxation in an LQ economy* and recursive optimal taxation, we study **complete-markets** models in which the government recognizes that it can manipulate Arrow securities prices.

Linear-quadratic versions of the Lucas-Stokey tax-smoothing model are described in *Optimal Taxation in an LQ Economy*.

That lecture is a warm-up for the non-linear-quadratic model of tax smoothing described in Optimal Taxation with State-Contingent Debt.

In both *Optimal Taxation in an LQ Economy* and Optimal Taxation with State-Contingent Debt, the government recognizes that its decisions affect prices.

In optimal taxation with incomplete markets, we study an **incomplete-markets** model in which the government also manipulates prices of government debt.

CHAPTER

THIRTEEN

MARKOV JUMP LINEAR QUADRATIC DYNAMIC PROGRAMMING

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

13.1 Overview

This lecture describes **Markov jump linear quadratic dynamic programming**, an extension of the method described in the first LQ control lecture.

Markov jump linear quadratic dynamic programming is described and analyzed in [Do Val *et al.*, 1999] and the references cited there.

The method has been applied to problems in macroeconomics and monetary economics by [Svensson *et al.*, 2008] and [Svensson and Williams, 2009].

The periodic models of seasonality described in chapter 14 of [Hansen and Sargent, 2013] are a special case of Markov jump linear quadratic problems.

Markov jump linear quadratic dynamic programming combines advantages of

- the computational simplicity of **linear quadratic dynamic programming**, with
- the ability of **finite state Markov chains** to represent interesting patterns of random variation.

The idea is to replace the constant matrices that define a **linear quadratic dynamic programming problem** with N sets of matrices that are fixed functions of the state of an N state Markov chain.

The state of the Markov chain together with the continuous $n \times 1$ state vector x_t form the state of the system.

For the class of infinite horizon problems being studied in this lecture, we obtain N interrelated matrix Riccati equations that determine N optimal value functions and N linear decision rules.

One of these value functions and one of these decision rules apply in each of the N Markov states.

That is, when the Markov state is in state j, the value function and the decision rule for state j prevails.

13.2 Review of useful LQ dynamic programming formulas

To begin, it is handy to have the following reminder in mind.

A **linear quadratic dynamic programming problem** consists of a scalar discount factor $\beta \in (0,1)$, an $n \times 1$ state vector x_t, an initial condition for x_0, a $k \times 1$ control vector u_t, a $p \times 1$ random shock vector w_{t+1} and the following two triples of matrices:

- A triple of matrices (R, Q, W) defining a loss function

$$r(x_t, u_t) = x_t' R x_t + u_t' Q u_t + 2 u_t' W x_t$$

- a triple of matrices (A, B, C) defining a state-transition law

$$x_{t+1} = A x_t + B u_t + C w_{t+1}$$

The problem is

$$-x_0' P x_0 - \rho = \min_{\{u_t\}_{t=0}^\infty} E \sum_{t=0}^{\infty} \beta^t r(x_t, u_t)$$

subject to the transition law for the state.

The optimal decision rule has the form

$$u_t = -F x_t$$

and the optimal value function is of the form

$$-\left(x_t' P x_t + \rho\right)$$

where P solves the algebraic matrix Riccati equation

$$P = R + \beta A' P A - (\beta B' P A + W)' (Q + \beta B P B)^{-1} (\beta B P A + W)$$

and the constant ρ satisfies

$$\rho = \beta \left(\rho + \text{trace}(P C C')\right)$$

and the matrix F in the decision rule for u_t satisfies

$$F = (Q + \beta B' P B)^{-1} (\beta (B' P A) + W)$$

With the preceding formulas in mind, we are ready to approach Markov Jump linear quadratic dynamic programming.

13.3 Linked Riccati equations for Markov LQ dynamic programming

The key idea is to make the matrices A, B, C, R, Q, W fixed functions of a finite state s that is governed by an N state Markov chain.

This makes decision rules depend on the Markov state, and so fluctuate through time in limited ways.

In particular, we use the following extension of a discrete-time linear quadratic dynamic programming problem.

We let $s_t \in [1, 2, \ldots, N]$ be a time t realization of an N-state Markov chain with transition matrix Π having typical element Π_{ij}.

Here i denotes today and j denotes tomorrow and

$$\Pi_{ij} = \text{Prob}(s_{t+1} = j | s_t = i)$$

We'll switch between labeling today's state as s_t and i and between labeling tomorrow's state as s_{t+1} or j.

The decision-maker solves the minimization problem:

$$\min_{\{u_t\}_{t=0}^{\infty}} E \sum_{t=0}^{\infty} \beta^t r(x_t, s_t, u_t)$$

with

$$r(x_t, s_t, u_t) = x_t' R_{s_t} x_t + u_t' Q_{s_t} u_t + 2 u_t' W_{s_t} x_t$$

subject to linear laws of motion with matrices (A, B, C) each possibly dependent on the Markov-state-s_t:

$$x_{t+1} = A_{s_t} x_t + B_{s_t} u_t + C_{s_t} w_{t+1}$$

where $\{w_{t+1}\}$ is an i.i.d. stochastic process with $w_{t+1} \sim N(0, I)$.

The optimal decision rule for this problem has the form

$$u_t = -F_{s_t} x_t$$

and the optimal value functions are of the form

$$-\left(x_t' P_{s_t} x_t + \rho_{s_t}\right)$$

or equivalently

$$-x_t' P_i x_t - \rho_i$$

The optimal value functions $-x' P_i x - \rho_i$ for $i = 1, \ldots, n$ satisfy the N interrelated Bellman equations

$$-x' P_i x - \rho_i = \max_u - \left[x' R_i x + u' Q_i u + 2u' W_i x + \beta \sum_j \Pi_{ij} E((A_i x + B_i u + C_i w)' P_j (A_i x + B_i u + C_i w) x + \rho_j) \right]$$

The matrices $P_{s_t} = P_i$ and the scalars $\rho_{s_t} = \rho_i, i = 1$, ..., n satisfy the following stacked system of **algebraic matrix Riccati** equations:

$$P_i = R_i + \beta \sum_j A_i' P_j A_i \Pi_{ij} - \sum_j \Pi_{ij} [(\beta B_i' P_j A_i + W_i)' (Q + \beta B_i' P_j B_i)^{-1} (\beta B_i' P_j A_i + W_i)]$$

$$\rho_i = \beta \sum_j \Pi_{ij} (\rho_j + \text{trace}(P_j C_i C_i'))$$

and the F_i in the optimal decision rules are

$$F_i = (Q_i + \beta \sum_j \Pi_{ij} B_i' P_j B_i)^{-1} (\beta \sum_j \Pi_{ij} (B_i' P_j A_i) + W_i)$$

13.4 Applications

We now describe some Python code and a few examples that put the code to work.

To begin, we import these Python modules

```
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
# Set discount factor
β = 0.95
```

13.5 Example 1

This example is a version of a classic problem of optimally adjusting a variable k_t to a target level in the face of costly adjustment.

This provides a model of gradual adjustment.

Given k_0, the objective function is

$$\max_{\{k_t\}_{t=1}^{\infty}} E_0 \sum_{t=0}^{\infty} \beta^t r\left(s_t, k_t\right)$$

where the one-period payoff function is

$$r(s_t, k_t) = f_{1,s_t} k_t - f_{2,s_t} k_t^2 - d_{s_t} (k_{t+1} - k_t)^2,$$

E_0 is a mathematical expectation conditioned on time 0 information x_0, s_0 and the transition law for continuous state variable k_t is

$$k_{t+1} - k_t = u_t$$

We can think of k_t as the decision-maker's capital and u_t as costs of adjusting the level of capital.

We assume that $f_1\left(s_t\right) > 0$, $f_2\left(s_t\right) > 0$, and $d\left(s_t\right) > 0$.

Denote the state transition matrix for Markov state $s_t \in \{1, 2\}$ as Π:

$$\Pr\left(s_{t+1} = j \mid s_t = i\right) = \Pi_{ij}$$

Let $x_t = \begin{bmatrix} k_t \\ 1 \end{bmatrix}$

We can represent the one-period payoff function $r\left(s_t, k_t\right)$ and the state-transition law as

$$\begin{aligned} r\left(s_t, k_t\right) &= f_{1,s_t} k_t - f_{2,s_t} k_t^2 - d_{s_t} {u_t}^2 \\ &= -x_t' \underbrace{\begin{bmatrix} f_{2,s_t} & -\frac{f_{1,s_t}}{2} \\ -\frac{f_{1,s_t}}{2} & 0 \end{bmatrix}}_{\equiv R(s_t)} x_t + \underbrace{d_{s_t}}_{\equiv Q(s_t)} {u_t}^2 \end{aligned}$$

$$x_{t+1} = \begin{bmatrix} k_{t+1} \\ 1 \end{bmatrix} = \underbrace{I_2}_{\equiv A(s_t)} x_t + \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\equiv B(s_t)} u_t$$

```python
def construct_arrays1(f1_vals=[1. ,1.],
                      f2_vals=[1., 1.],
                      d_vals=[1., 1.]):
    """
    Construct matrices that map the problem described in example 1
    into a Markov jump linear quadratic dynamic programming problem
    """

    # Number of Markov states
    m = len(f1_vals)
    # Number of state and control variables
    n, k = 2, 1

    # Construct sets of matrices for each state
    As = [np.eye(n) for i in range(m)]
    Bs = [np.array([[1, 0]]).T for i in range(m)]

    Rs = np.zeros((m, n, n))
    Qs = np.zeros((m, k, k))

    for i in range(m):
        Rs[i, 0, 0] = f2_vals[i]
        Rs[i, 1, 0] = - f1_vals[i] / 2
        Rs[i, 0, 1] = - f1_vals[i] / 2

        Qs[i, 0, 0] = d_vals[i]

    Cs, Ns = None, None

    # Compute the optimal k level of the payoff function in each state
    k_star = np.empty(m)
    for i in range(m):
        k_star[i] = f1_vals[i] / (2 * f2_vals[i])

    return Qs, Rs, Ns, As, Bs, Cs, k_star
```

The continuous part of the state x_t consists of two variables, namely, k_t and a constant term.

```python
state_vec1 = ["k", "constant term"]
```

We start with a Markov transition matrix that makes the Markov state be strictly periodic:

$$\Pi_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

We set f_{1,s_t} and f_{2,s_t} to be independent of the Markov state s_t

$$f_{1,1} = f_{1,2} = 1,$$

$$f_{2,1} = f_{2,2} = 1$$

In contrast to f_{1,s_t} and f_{2,s_t}, we make the adjustment cost d_{s_t} vary across Markov states s_t.

We set the adjustment cost to be lower in Markov state 2

$$d_1 = 1, d_2 = 0.5$$

The following code forms a Markov switching LQ problem and computes the optimal value functions and optimal decision rules for each Markov state

```
# Construct Markov transition matrix
Π1 = np.array([[0., 1.],
               [1., 0.]])
```

```
# Construct matrices
Qs, Rs, Ns, As, Bs, Cs, k_star = construct_arrays1(d_vals=[1., 0.5])
```

```
# Construct a Markov Jump LQ problem
ex1_a = qe.LQMarkov(Π1, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
# Solve for optimal value functions and decision rules
ex1_a.stationary_values();
```

Let's look at the value function matrices and the decision rules for each Markov state

```
# P(s)
ex1_a.Ps
```

```
array([[[ 1.56626026, -0.78313013],
        [-0.78313013, -4.60843493]],

       [[ 1.37424214, -0.68712107],
        [-0.68712107, -4.65643947]]])
```

```
# d(s) = 0, since there is no randomness
ex1_a.ds
```

```
array([0., 0.])
```

```
# F(s)
ex1_a.Fs
```

```
array([[[ 0.56626026, -0.28313013]],

       [[ 0.74848427, -0.37424214]]])
```

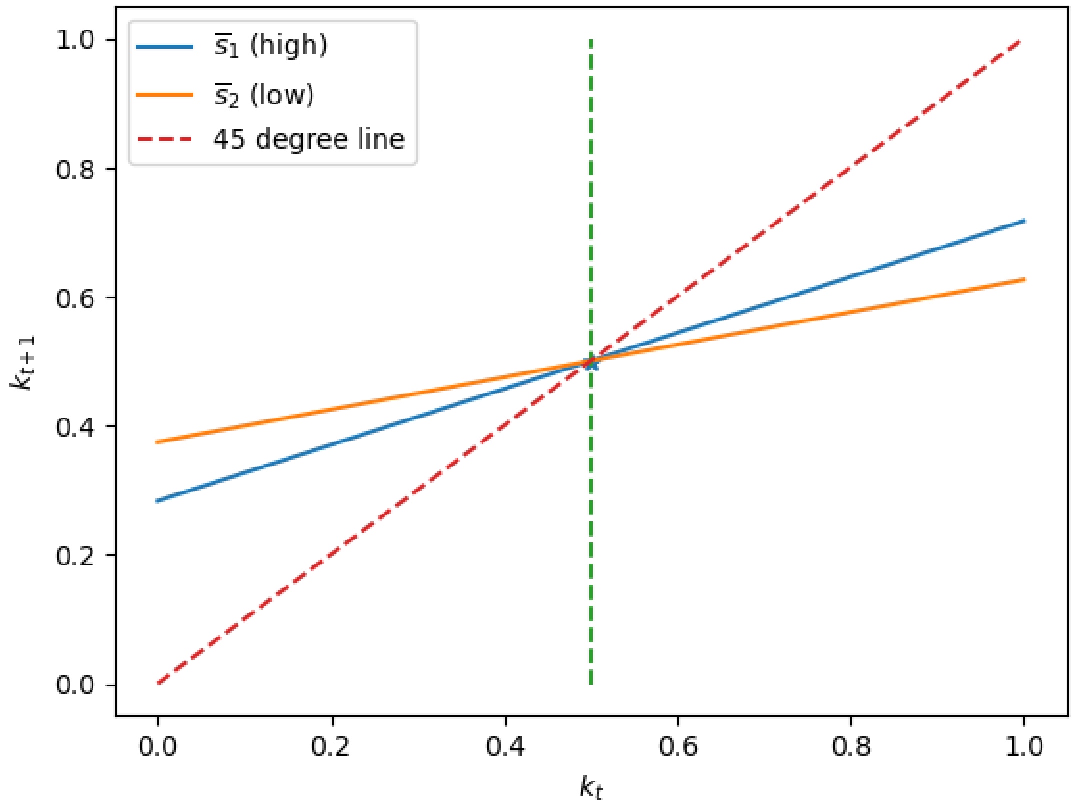
Now we'll plot the decision rules and see if they make sense

```
# Plot the optimal decision rules
k_grid = np.linspace(0., 1., 100)
# Optimal choice in state s1
u1_star = - ex1_a.Fs[0, 0, 1] - ex1_a.Fs[0, 0, 0] * k_grid
# Optimal choice in state s2
u2_star = - ex1_a.Fs[1, 0, 1] - ex1_a.Fs[1, 0, 0] * k_grid

fig, ax = plt.subplots()
ax.plot(k_grid, k_grid + u1_star, label="$\overline{s}_1$ (high)")
ax.plot(k_grid, k_grid + u2_star, label="$\overline{s}_2$ (low)")

# The optimal k*
ax.scatter([0.5, 0.5], [0.5, 0.5], marker="*")
ax.plot([k_star[0], k_star[0]], [0., 1.0], '--')
```

(continues on next page)

(continued from previous page)

```
# 45 degree line
ax.plot([0., 1.], [0., 1.], '--', label="45 degree line")

ax.set_xlabel("$k_t$")
ax.set_ylabel("$k_{t+1}$")
ax.legend()
plt.show()
```



The above graph plots $k_{t+1} = k_t + u_t = k_t - Fx_t$ as an affine (i.e., linear in $k_t$ plus a constant) function of $k_t$ for both Markov states s_t.

It also plots the 45 degree line.

Notice that the two s_t-dependent *closed loop* functions that determine $k_{t+1}$ as functions of $k_t$ share the same rest point (also called a fixed point) at $k_t = 0.5$.

Evidently, the optimal decision rule in Markov state 2, in which the adjustment cost is lower, makes $k_{t+1}$ a flatter function of $k_t$ in Markov state 2.

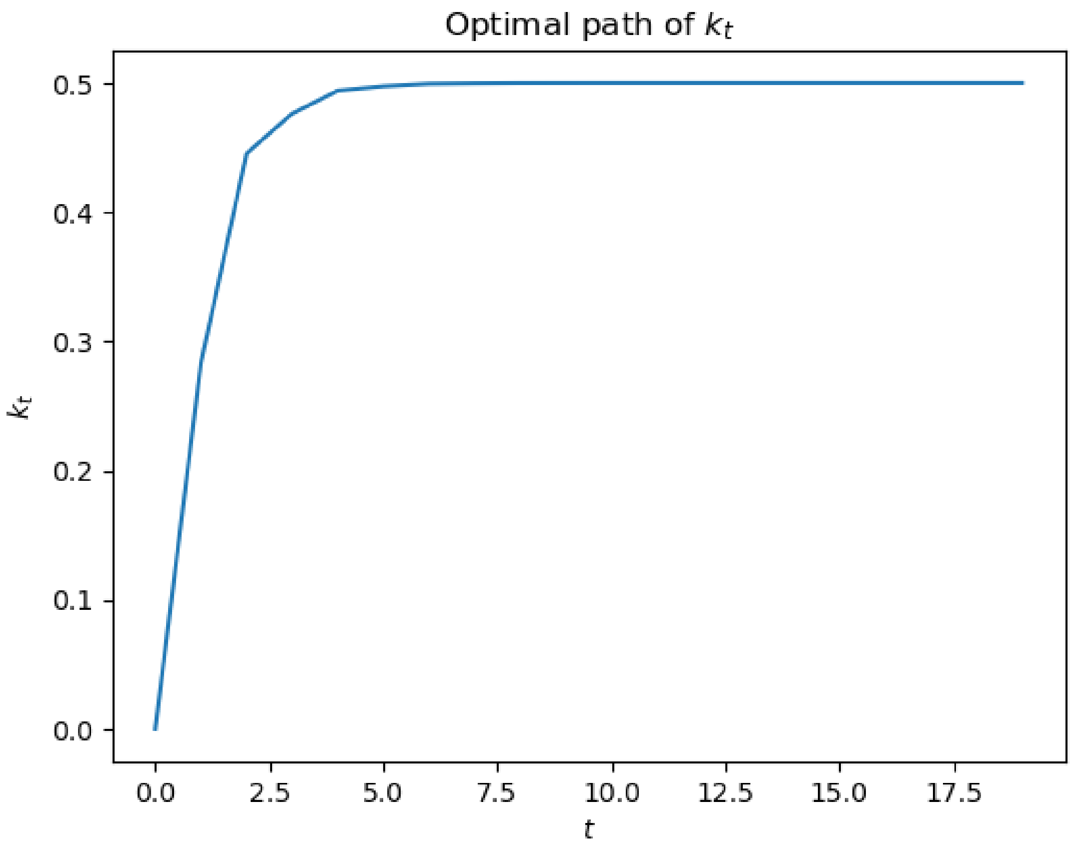
This happens because when $k_t$ is not at its fixed point, $|u_{t,2}| > |u_{t,2}|$, so that the decision-maker adjusts toward the fixed point faster when the Markov state s_t takes a value that makes it cheaper.

```
# Compute time series
T = 20
x0 = np.array([[0., 1.]]).T
x_path = ex1_a.compute_sequence(x0, ts_length=T)[0]
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots()
ax.plot(range(T), x_path[0, :-1])
ax.set_xlabel("$t$")
ax.set_ylabel("$k_t$")
ax.set_title("Optimal path of $k_t$")
plt.show()
```



Now we'll depart from the preceding transition matrix that made the Markov state be strictly periodic.

We'll begin with symmetric transition matrices of the form

$$\Pi_2 = \begin{bmatrix} 1-\lambda & \lambda \\ \lambda & 1-\lambda \end{bmatrix}.$$

```
λ = 0.8 # high λ
Π2 = np.array([[1-λ, λ],
               [λ, 1-λ]])

ex1_b = qe.LQMarkov(Π2, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
ex1_b.stationary_values();
ex1_b.Fs
```

```
array([[[ 0.57291724, -0.28645862]],

       [[ 0.74434525, -0.37217263]]])
```

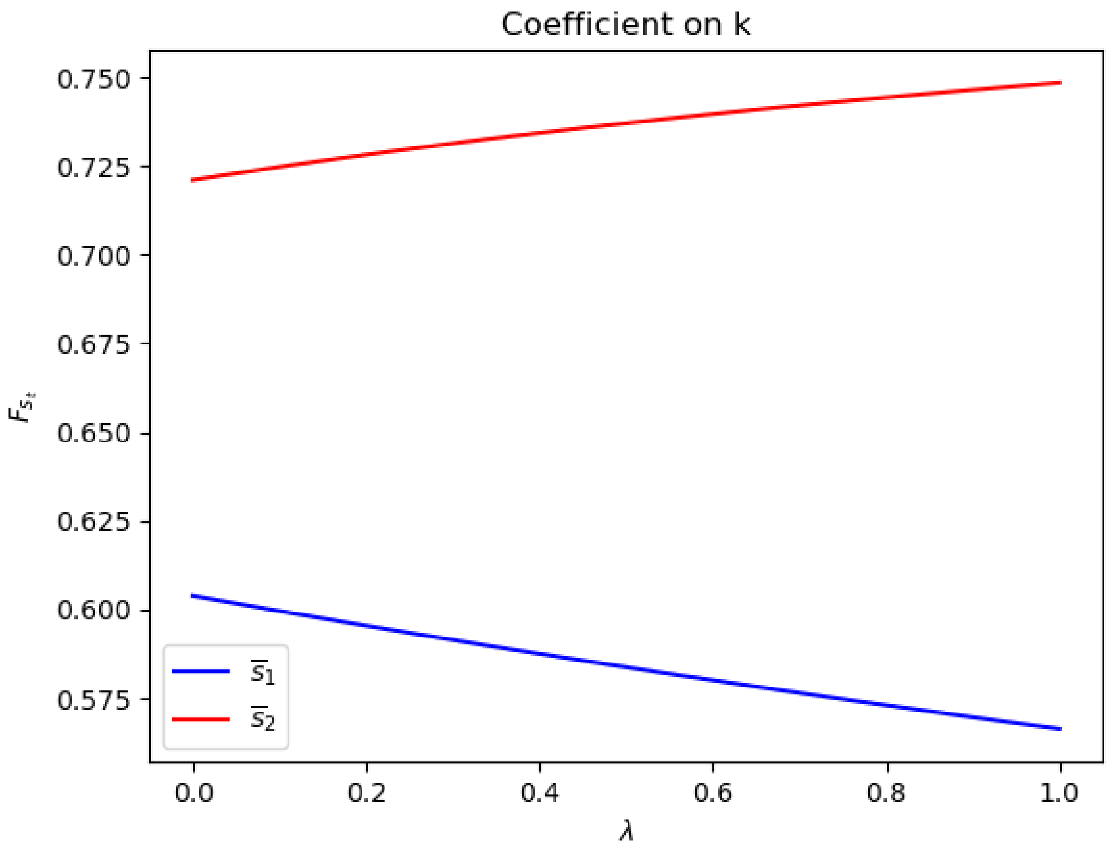
```
λ = 0.2 # low λ
Π2 = np.array([[1-λ, λ],
               [λ, 1-λ]])

ex1_b = qe.LQMarkov(Π2, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
ex1_b.stationary_values();
ex1_b.Fs
```

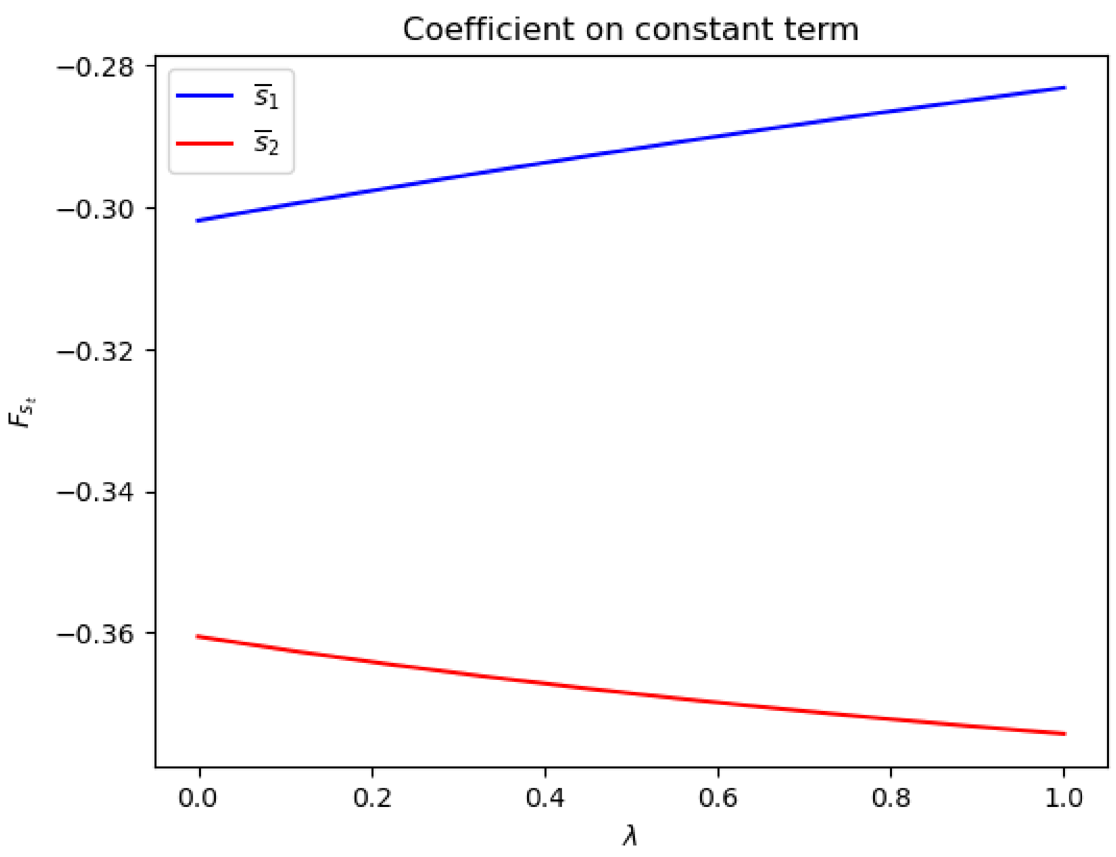
```
array([[[ 0.59533259, -0.2976663 ]],

       [[ 0.72818728, -0.36409364]]])
```

We can plot optimal decision rules associated with different λ values.

```
λ_vals = np.linspace(0., 1., 10)
F1 = np.empty((λ_vals.size, 2))
F2 = np.empty((λ_vals.size, 2))

for i, λ in enumerate(λ_vals):
    Π2 = np.array([[1-λ, λ],
                   [λ, 1-λ]])

    ex1_b = qe.LQMarkov(Π2, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
    ex1_b.stationary_values();
    F1[i, :] = ex1_b.Fs[0, 0, :]
    F2[i, :] = ex1_b.Fs[1, 0, :]
```

```
for i, state_var in enumerate(state_vec1):
    fig, ax = plt.subplots()
    ax.plot(λ_vals, F1[:, i], label="$\overline{s}_1$", color="b")
    ax.plot(λ_vals, F2[:, i], label="$\overline{s}_2$", color="r")

    ax.set_xlabel("$\lambda$")
    ax.set_ylabel("$F_{s_t}$")
    ax.set_title(f"Coefficient on {state_var}")
    ax.legend()
    plt.show()
```


Coefficient on k
F_{s_t}
λ
$\overline{s}_1$
$\overline{s}_2$
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0.575
0.0
0.2
0.4
0.6
0.8
1.0



Notice how the decision rules' constants and slopes behave as functions of λ.

Evidently, as the Markov chain becomes *more nearly periodic* (i.e., as $\lambda \to 1$), the dynamic program adjusts capital faster in the low adjustment cost Markov state to take advantage of what is only temporarily a more favorable time to invest.

Now let's study situations in which the Markov transition matrix Π is asymmetric

$$\Pi_3 = \begin{bmatrix} 1-\lambda & \lambda \\ \delta & 1-\delta \end{bmatrix}.$$

```
λ, δ = 0.8, 0.2
Π3 = np.array([[1-λ, λ],
               [δ, 1-δ]])

ex1_b = qe.LQMarkov(Π3, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
ex1_b.stationary_values();
ex1_b.Fs
```

```
array([[[ 0.57169781, -0.2858489 ]],

       [[ 0.72749075, -0.36374537]]])
```

We can plot optimal decision rules for different λ and δ values.

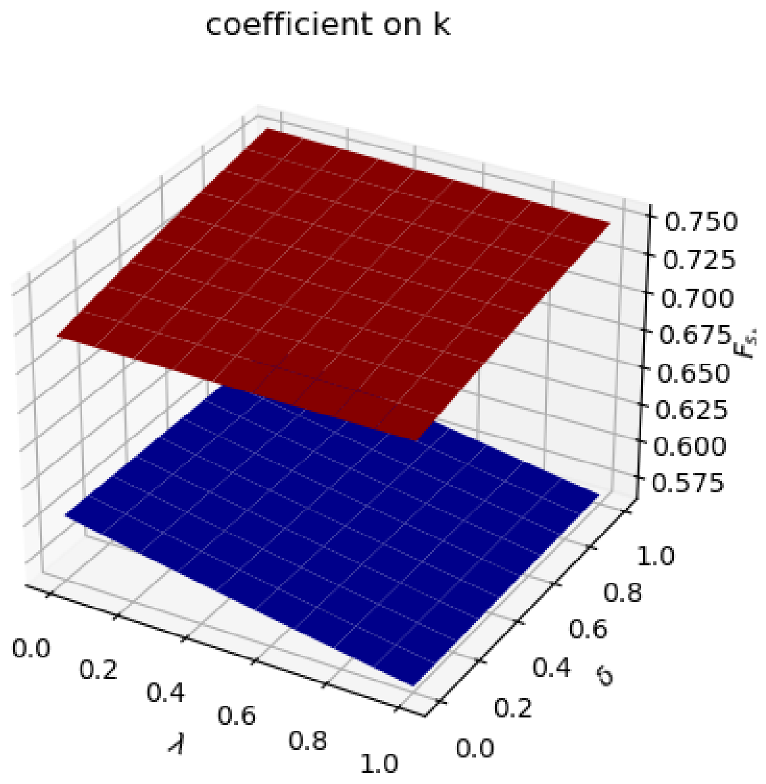
```
λ_vals = np.linspace(0., 1., 10)
δ_vals = np.linspace(0., 1., 10)

```

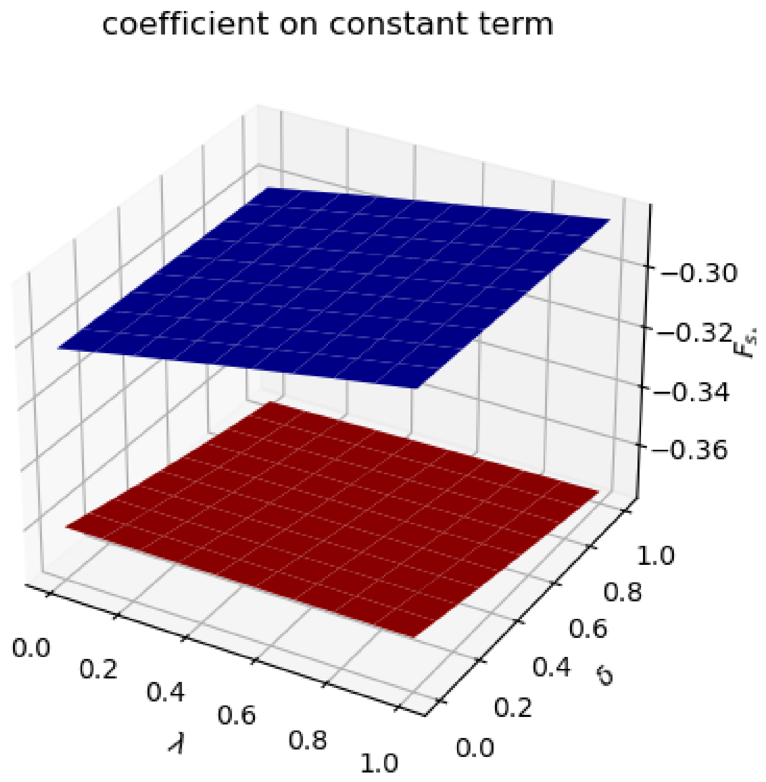
(continues on next page)

(continued from previous page)

```
λ_grid = np.empty((λ_vals.size, δ_vals.size))
δ_grid = np.empty((λ_vals.size, δ_vals.size))
F1_grid = np.empty((λ_vals.size, δ_vals.size, len(state_vec1)))
F2_grid = np.empty((λ_vals.size, δ_vals.size, len(state_vec1)))

for i, λ in enumerate(λ_vals):
    λ_grid[i, :] = λ
    δ_grid[i, :] = δ_vals
    for j, δ in enumerate(δ_vals):
        Π3 = np.array([[1-λ, λ],
                       [δ, 1-δ]])

        ex1_b = qe.LQMarkov(Π3, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
        ex1_b.stationary_values();
        F1_grid[i, j, :] = ex1_b.Fs[0, 0, :]
        F2_grid[i, j, :] = ex1_b.Fs[1, 0, :]
```

```
for i, state_var in enumerate(state_vec1):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    # high adjustment cost, blue surface
    ax.plot_surface(λ_grid, δ_grid, F1_grid[:, :, i], color="b")
    # low adjustment cost, red surface
    ax.plot_surface(λ_grid, δ_grid, F2_grid[:, :, i], color="r")
    ax.set_xlabel("$\lambda$")
    ax.set_ylabel("$\delta$")
    ax.set_zlabel("$F_{s_t}$")
    ax.set_title(f"coefficient on {state_var}")
    plt.show()
```





The following code defines a wrapper function that computes optimal decision rules for cases with different Markov transition matrices

```
def run(construct_func, vals_dict, state_vec):
    """
    A Wrapper function that repeats the computation above
    for different cases
    """

    Qs, Rs, Ns, As, Bs, Cs, k_star = construct_func(**vals_dict)

    # Symmetric Π
    # Notice that pure periodic transition is a special case
    # when λ=1
    print("symmetric Π case:\n")
    λ_vals = np.linspace(0., 1., 10)
    F1 = np.empty((λ_vals.size, len(state_vec)))
    F2 = np.empty((λ_vals.size, len(state_vec)))

    for i, λ in enumerate(λ_vals):
        Π2 = np.array([[1-λ, λ],
                       [λ, 1-λ]])

        mplq = qe.LQMarkov(Π2, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
        mplq.stationary_values();
        F1[i, :] = mplq.Fs[0, 0, :]
        F2[i, :] = mplq.Fs[1, 0, :]

    for i, state_var in enumerate(state_vec):
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(λ_vals, F1[:, i], label="$\overline{s}_1$", color="b")
        ax.plot(λ_vals, F2[:, i], label="$\overline{s}_2$", color="r")

        ax.set_xlabel("$\lambda$")
        ax.set_ylabel("$F(\overline{s}_t)$")
        ax.set_title(f"coefficient on {state_var}")
        ax.legend()
        plt.show()

    # Plot optimal k*_{s_t} and k that optimal policies are targeting
    # only for example 1
    if state_vec == ["k", "constant term"]:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        for i in range(2):
            F = [F1, F2][i]
            c = ["b", "r"][i]
            ax.plot([0, 1], [k_star[i], k_star[i]], "--",
                    color=c, label="$k^*(\overline{s}_"+str(i+1)+")$")
            ax.plot(λ_vals, - F[:, 1] / F[:, 0], color=c,
                    label="$k^{target}(\overline{s}_"+str(i+1)+")$")

        # Plot a vertical line at λ=0.5
        ax.plot([0.5, 0.5], [min(k_star), max(k_star)], "-.")

        ax.set_xlabel("$\lambda$")
        ax.set_ylabel("$k$")
        ax.set_title("Optimal k levels and k targets")
        ax.text(0.5, min(k_star)+(max(k_star)-min(k_star))/20, "$\lambda=0.5$")
```

(continues on next page)

(continued from previous page)

```
        ax.legend(bbox_to_anchor=(1., 1.))
        plt.show()

    # Asymmetric Π
    print("asymmetric Π case:\n")
    δ_vals = np.linspace(0., 1., 10)

    λ_grid = np.empty((λ_vals.size, δ_vals.size))
    δ_grid = np.empty((λ_vals.size, δ_vals.size))
    F1_grid = np.empty((λ_vals.size, δ_vals.size, len(state_vec)))
    F2_grid = np.empty((λ_vals.size, δ_vals.size, len(state_vec)))

    for i, λ in enumerate(λ_vals):
        λ_grid[i, :] = λ
        δ_grid[i, :] = δ_vals
        for j, δ in enumerate(δ_vals):
            Π3 = np.array([[1-λ, λ],
                           [δ, 1-δ]])

            mplq = qe.LQMarkov(Π3, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
            mplq.stationary_values();
            F1_grid[i, j, :] = mplq.Fs[0, 0, :]
            F2_grid[i, j, :] = mplq.Fs[1, 0, :]

    for i, state_var in enumerate(state_vec):
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot_surface(λ_grid, δ_grid, F1_grid[:, :, i], color="b")
        ax.plot_surface(λ_grid, δ_grid, F2_grid[:, :, i], color="r")
        ax.set_xlabel("$\lambda$")
        ax.set_ylabel("$\delta$")
        ax.set_zlabel("$F(\overline{s}_t)$")
        ax.set_title(f"coefficient on {state_var}")
        plt.show()
```

To illustrate the code with another example, we shall set f_{2,s_t} and d_{s_t} as constant functions and

$$f_{1,1} = 0.5, f_{1,2} = 1$$

Thus, the sole role of the Markov jump state s_t is to identify times in which capital is very productive and other times in which it is less productive.

The example below reveals much about the structure of the optimum problem and optimal policies.

Only f_{1,s_t} varies with s_t.

So there are different s_t-dependent optimal static k level in different states $k^*_{s_t} = \frac{f_{1,s_t}}{2f_{2,s_t}}$, values of k that maximize one-period payoff functions in each state.

We denote a target k level as $k^{target}_{s_t}$, the fixed point of the optimal policies in each state, given the value of $\lambda$.

We call $k^{target}_{s_t}$ a "target" because in each Markov state s_t, optimal policies are contraction mappings and will push k_t towards a fixed point $k^{target}_{s_t}$.

When $\lambda \to 0$, each Markov state becomes close to absorbing state and consequently $k^{target}_{s_t} \to k^*_{s_t}$.

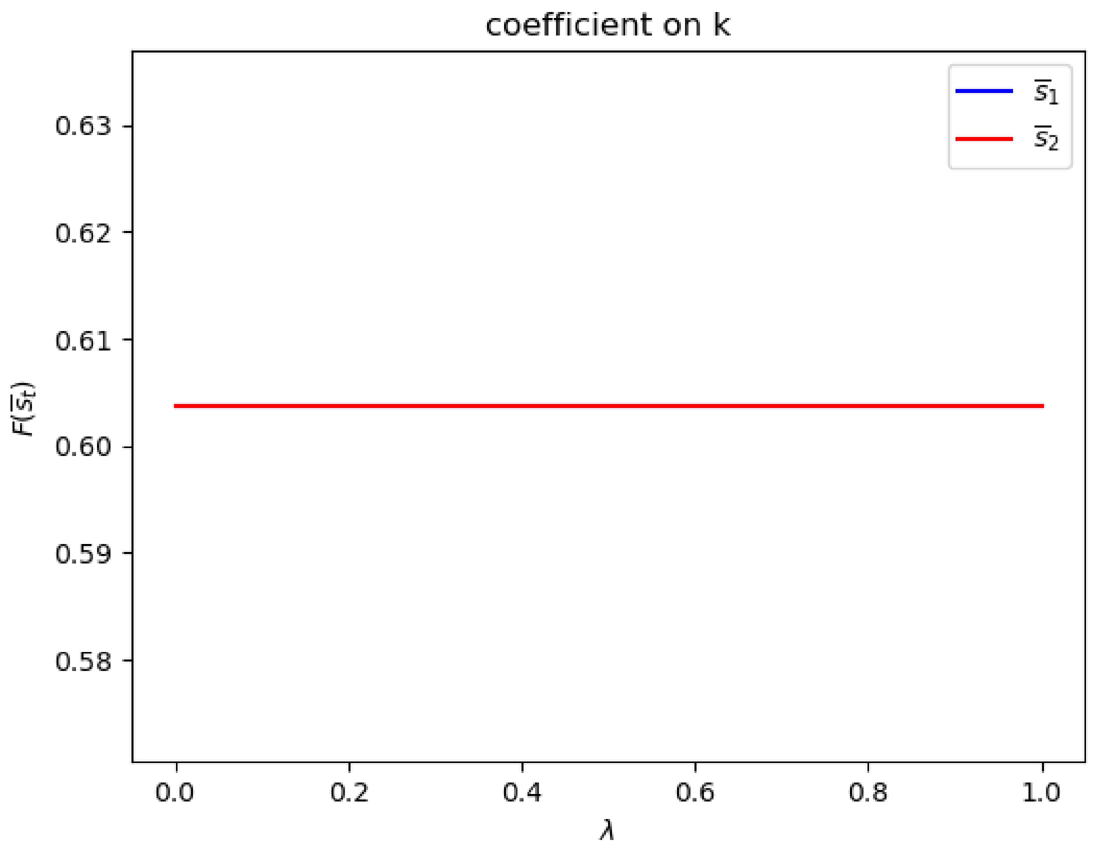
But when $\lambda \to 1$, the Markov transition matrix becomes more nearly periodic, so the optimum decision rules target more at the optimal k level in the other state in order to enjoy higher expected payoff in the next period.

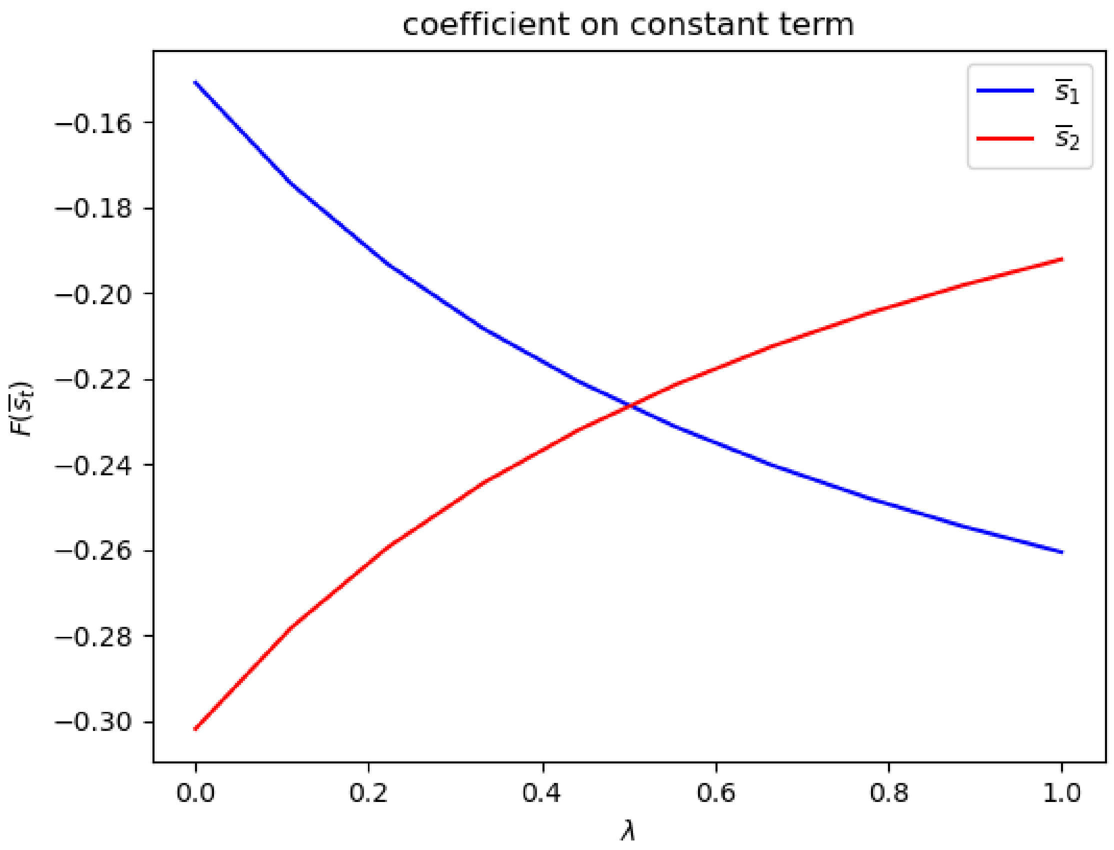
The switch happens at $\lambda = 0.5$ when both states are equally likely to be reached.

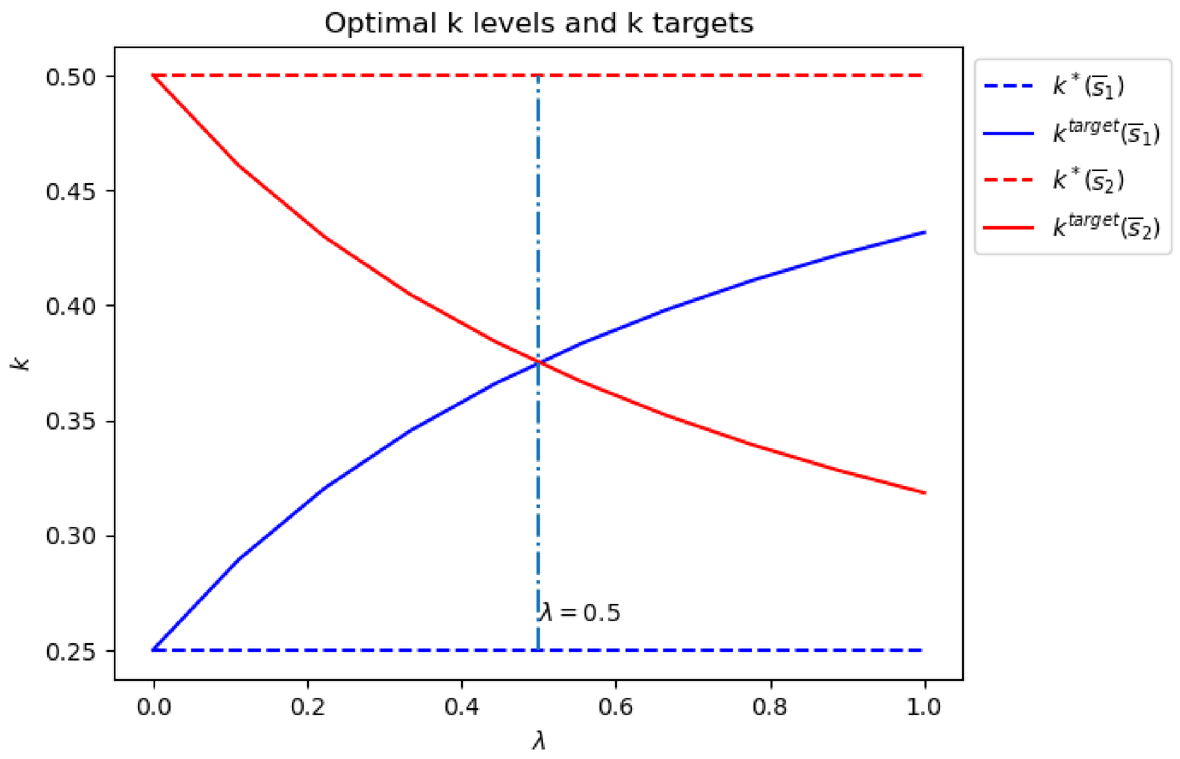
Below we plot an additional figure that shows optimal k levels in the two states Markov jump state and also how the targeted k levels change as λ changes.

```
run(construct_arrays1, {"f1_vals":[0.5, 1.]}, state_vec1)
```

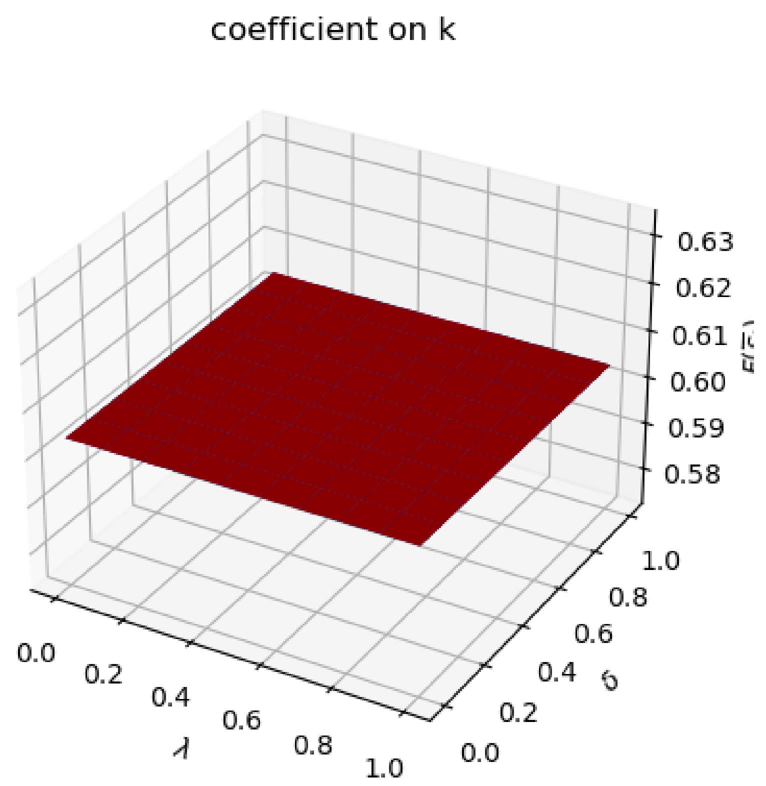
```
symmetric Π case:
```

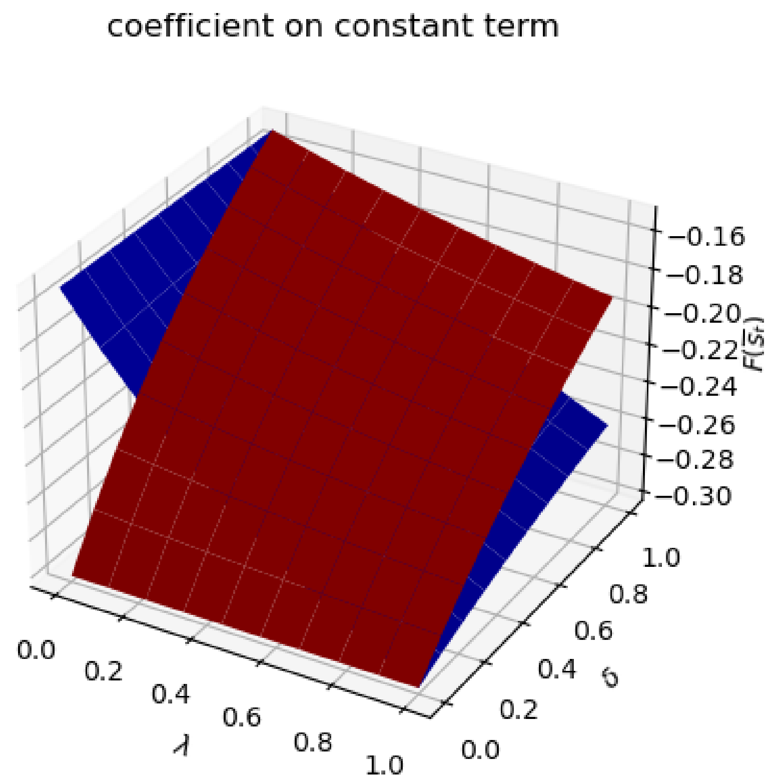



coefficient on constant term
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
−0.16
−0.18
−0.20
−0.22
−0.24
−0.26
−0.28
−0.30
0.0
0.2
0.4
0.6
0.8
1.0
λ


Optimal k levels and k targets
$k^*(\overline{s}_1)$
$k^{target}(\overline{s}_1)$
$k^*(\overline{s}_2)$
$k^{target}(\overline{s}_2)$
k
0.50
0.45
0.40
0.35
0.30
0.25
$\lambda = 0.5$
0.0
0.2
0.4
0.6
0.8
1.0
λ

```
asymmetric Π case:
```



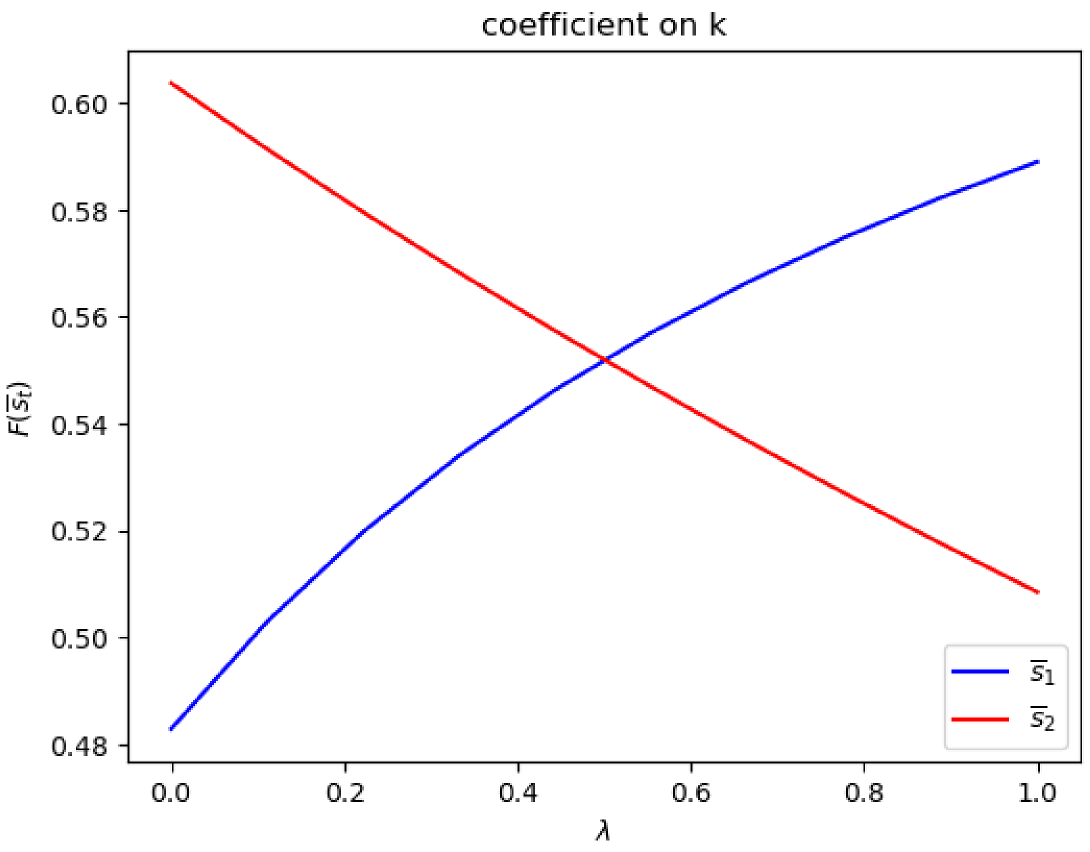


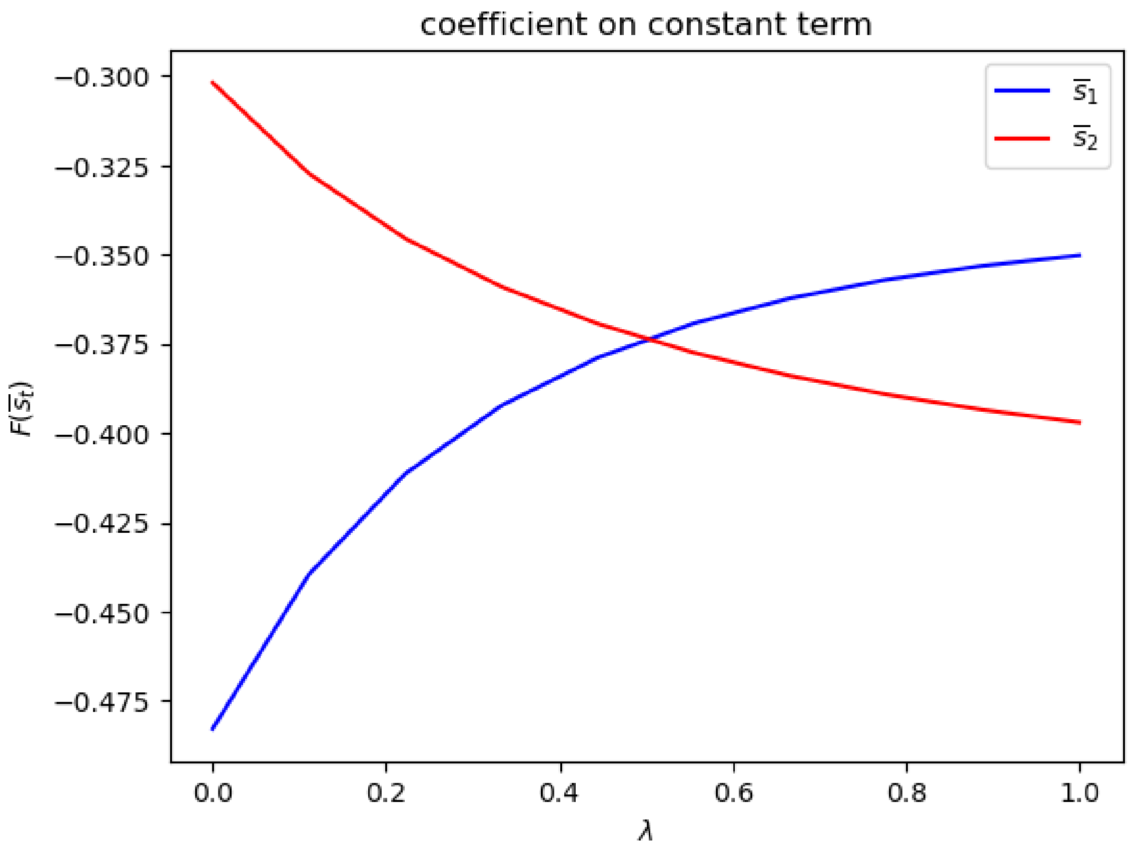
Set f_{1,s_t} and d_{s_t} as constant functions and

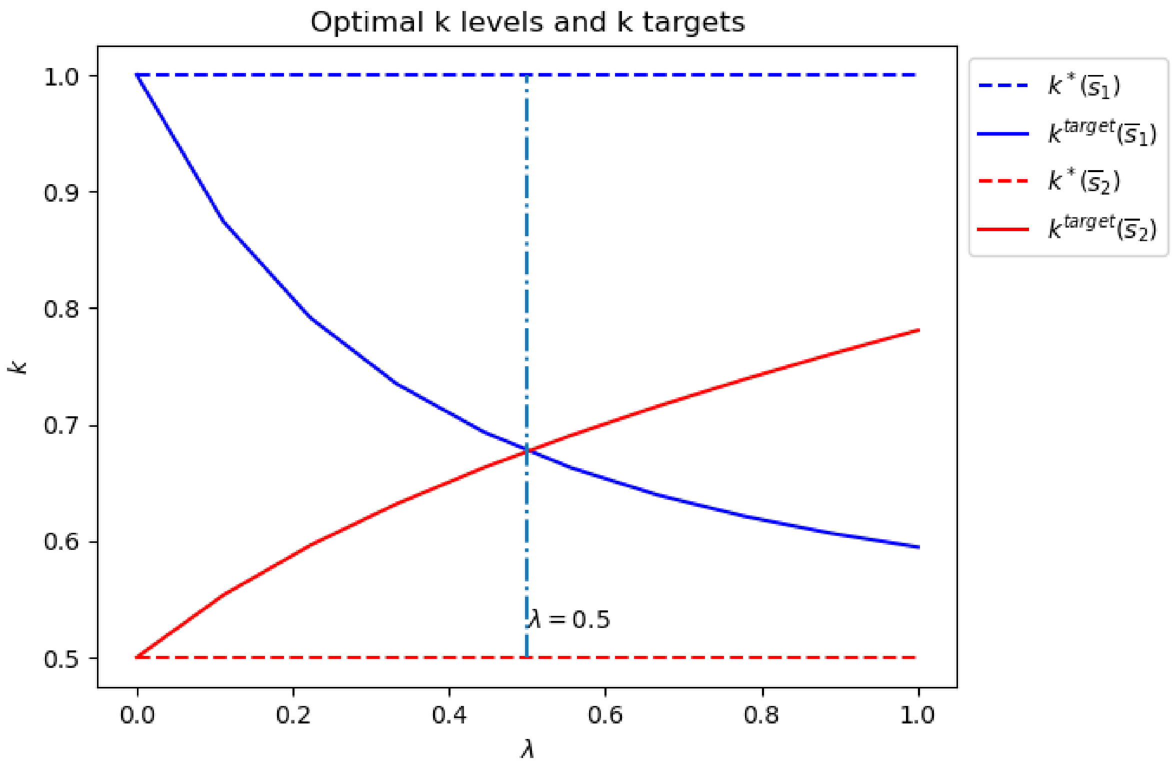
$$f_{2,1} = 0.5, f_{2,2} = 1$$

```
run(construct_arrays1, {"f2_vals":[0.5, 1.]}, state_vec1)
```

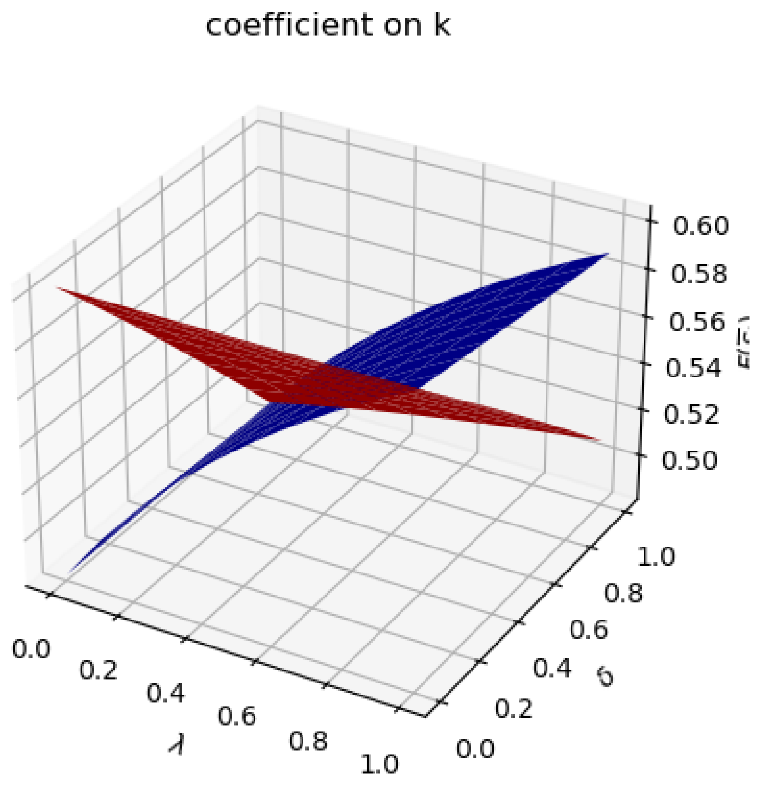
```
symmetric Π case:
```

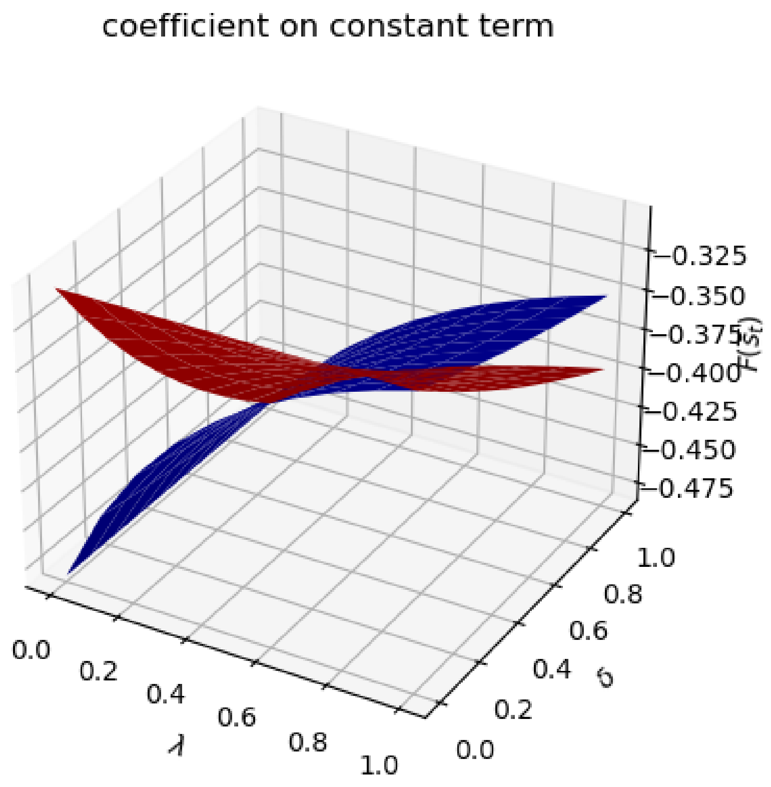



coefficient on constant term
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
λ


Optimal k levels and k targets
$k^*(\overline{s}_1)$
$k^{target}(\overline{s}_1)$
$k^*(\overline{s}_2)$
$k^{target}(\overline{s}_2)$
$\lambda = 0.5$
k
λ

```
asymmetric Π case:
```





13.6 Example 2

We now add to the example 1 setup another state variable w_t that follows the evolution law

$$w_{t+1} = \alpha_0 (s_t) + \rho (s_t) w_t + \sigma (s_t) \epsilon_{t+1}, \quad \epsilon_{t+1} \sim N(0,1)$$

We think of w_t as a rental rate or tax rate that the decision maker pays each period for k_t.

To capture this idea, we add to the decision-maker's one-period payoff function the product of w_t and k_t

$$r(s_t, k_t, w_t) = f_{1,s_t} k_t - f_{2,s_t} k_t^2 - d_{s_t} (k_{t+1} - k_t)^2 - w_t k_t,$$

We now let the continuous part of the state at time t be $x_t = \begin{bmatrix} k_t \\ 1 \\ w_t \end{bmatrix}$ and continue to set the control $u_t = k_{t+1} - k_t$.

We can write the one-period payoff function $r(s_t, k_t, w_t)$ and the state-transition law as

$$\begin{aligned} r(s_t, k_t, w_t) &= f_1 (s_t) k_t - f_2 (s_t) k_t^2 - d(s_t) (k_{t+1} - k_t)^2 - w_t k_t \\ &= - \left(x_t' \underbrace{\begin{bmatrix} f_2(s_t) & -\frac{f_1(s_t)}{2} & \frac{1}{2} \\ -\frac{f_1(s_t)}{2} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix}}_{\equiv R(s_t)} x_t + \underbrace{d(s_t)}_{\equiv Q(s_t)} u_t^2 \right), \end{aligned}$$

and

$$x_{t+1} = \begin{bmatrix} k_{t+1} \\ 1 \\ w_{t+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \alpha_0 (s_t) & \rho (s_t) \end{bmatrix}}_{\equiv A(s_t)} x_t + \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{\equiv B(s_t)} u_t + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \sigma (s_t) \end{bmatrix}}_{\equiv C(s_t)} \epsilon_{t+1}$$

```python
def construct_arrays2(f1_vals=[1. ,1.],
                      f2_vals=[1., 1.],
                      d_vals=[1., 1.],
                      α0_vals=[1., 1.],
                      ρ_vals=[0.9, 0.9],
                      σ_vals=[1., 1.]):
    """
    Construct matrices that maps the problem described in example 2
    into a Markov jump linear quadratic dynamic programming problem.
    """

    m = len(f1_vals)
    n, k, j = 3, 1, 1

    Rs = np.zeros((m, n, n))
    Qs = np.zeros((m, k, k))
    As = np.zeros((m, n, n))
    Bs = np.zeros((m, n, k))
    Cs = np.zeros((m, n, j))

    for i in range(m):
        Rs[i, 0, 0] = f2_vals[i]
        Rs[i, 1, 0] = - f1_vals[i] / 2
```

(continues on next page)

(continued from previous page)

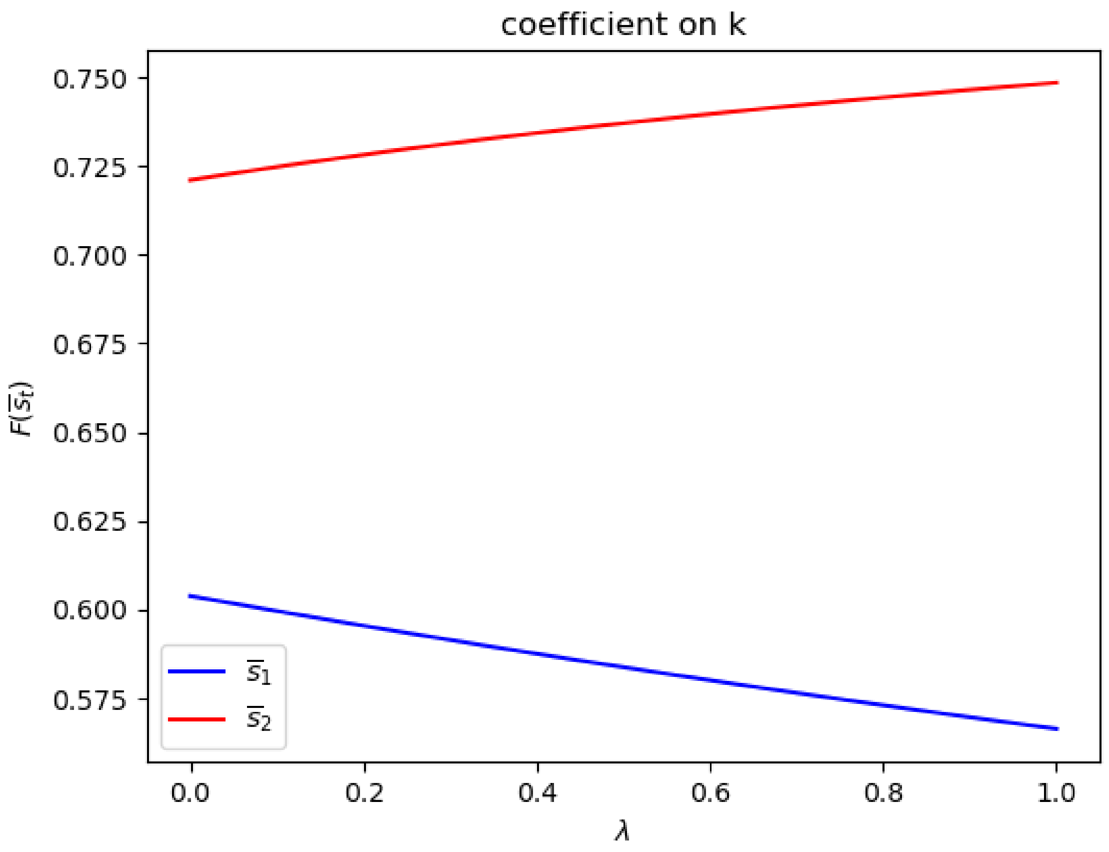
```
        Rs[i, 0, 1] = - f1_vals[i] / 2
        Rs[i, 0, 2] = 1/2
        Rs[i, 2, 0] = 1/2

        Qs[i, 0, 0] = d_vals[i]

        As[i, 0, 0] = 1
        As[i, 1, 1] = 1
        As[i, 2, 1] = α0_vals[i]
        As[i, 2, 2] = ρ_vals[i]

        Bs[i, :, :] = np.array([[1, 0, 0]]).T

        Cs[i, :, :] = np.array([[0, 0, σ_vals[i]]]).T

    Ns = None
    k_star = None

    return Qs, Rs, Ns, As, Bs, Cs, k_star
```

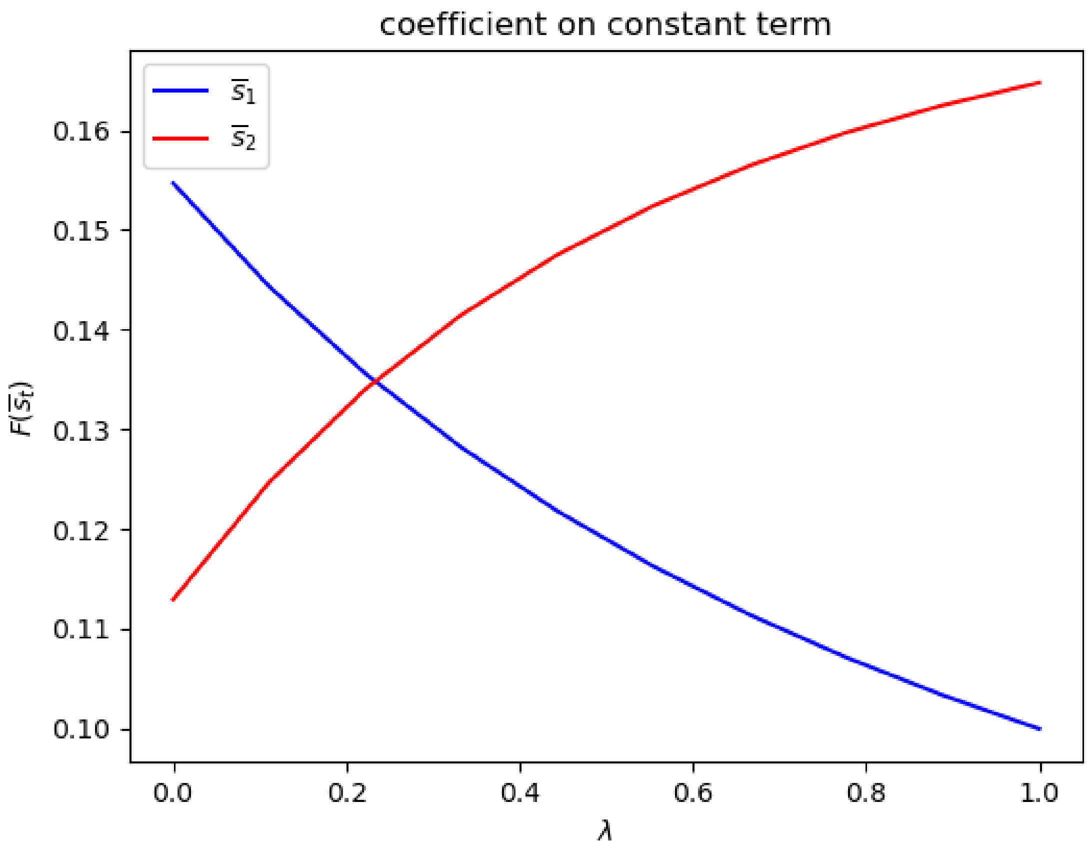
```
state_vec2 = ["k", "constant term", "w"]
```

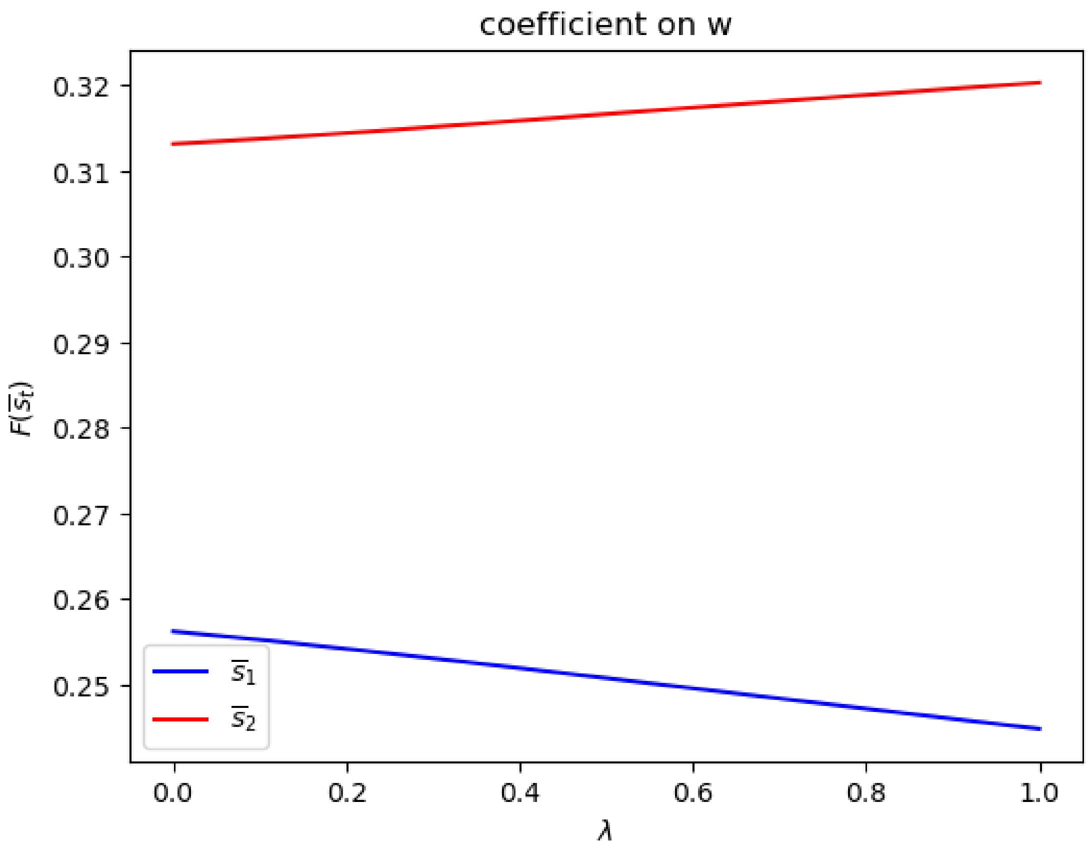
Only d_{s_t} depends on s_t.

```
run(construct_arrays2, {"d_vals":[1., 0.5]}, state_vec2)
```

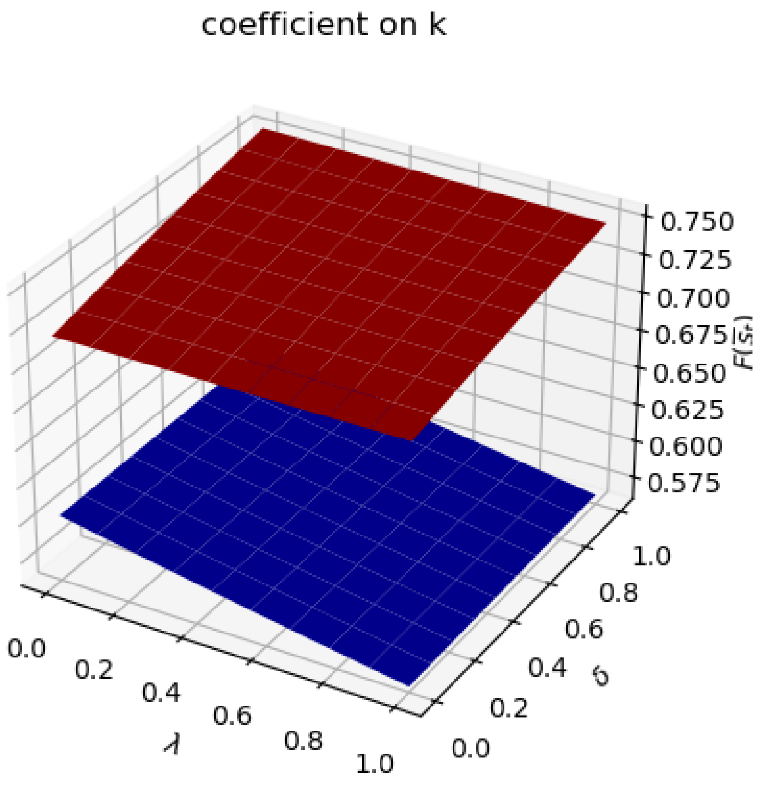
```
symmetric Π case:
```

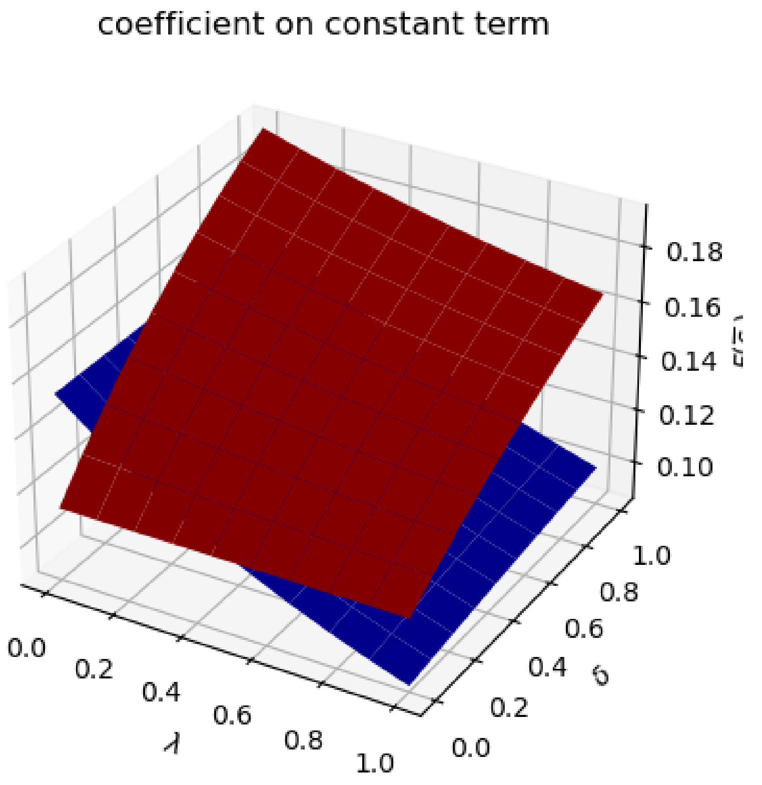

coefficient on k
$F(\overline{s}_t)$
λ
$\overline{s}_1$
$\overline{s}_2$
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0.575
0.0
0.2
0.4
0.6
0.8
1.0

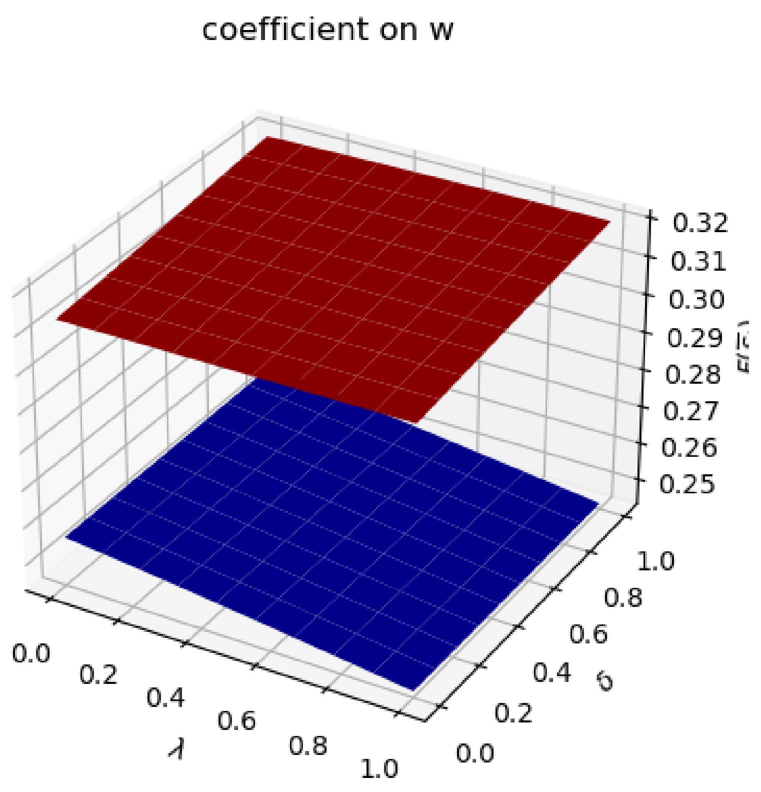

coefficient on constant term
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
0.10
0.11
0.12
0.13
0.14
0.15
0.16
0.0
0.2
0.4
0.6
0.8
1.0
λ



```
asymmetric Π case:
```


coefficient on k
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0.575
F(S_t)
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
δ

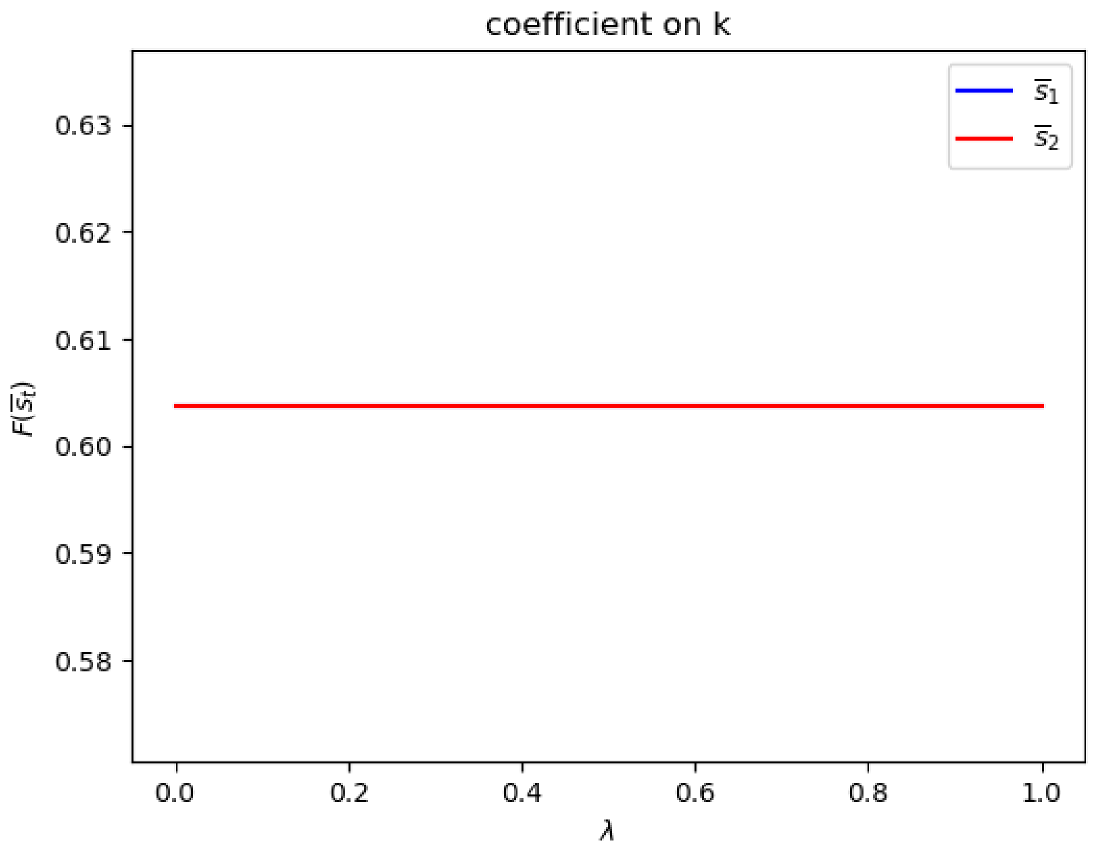

coefficient on constant term
0.18
0.16
0.14
0.12
0.10
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
δ

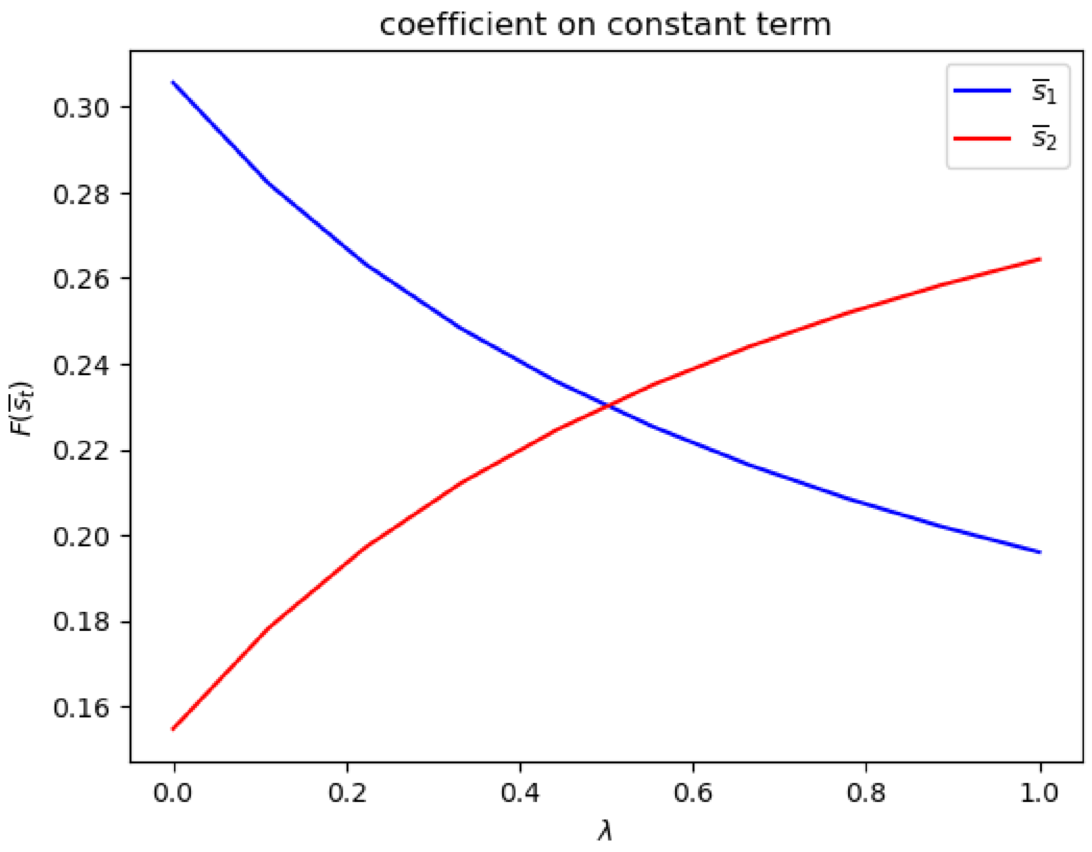


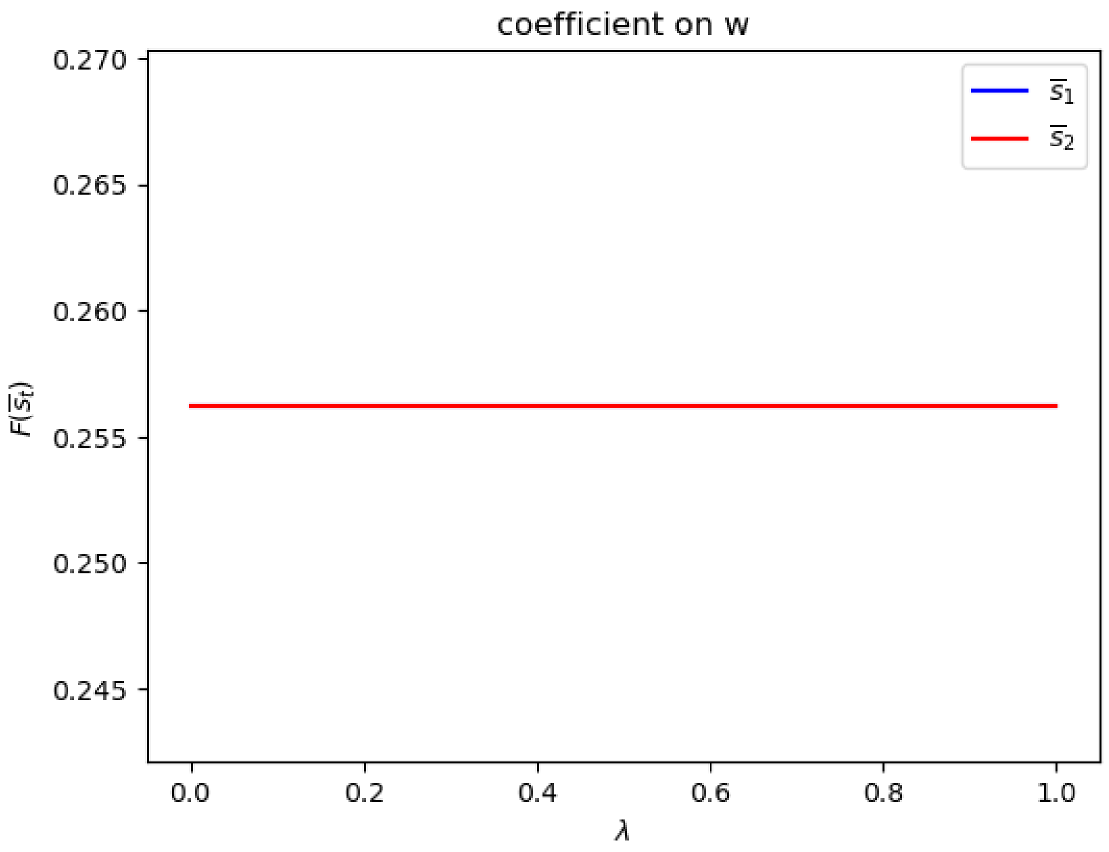
Only f_{1,s_t} depends on s_t.

```
run(construct_arrays2, {"f1_vals":[0.5, 1.]}, state_vec2)
```

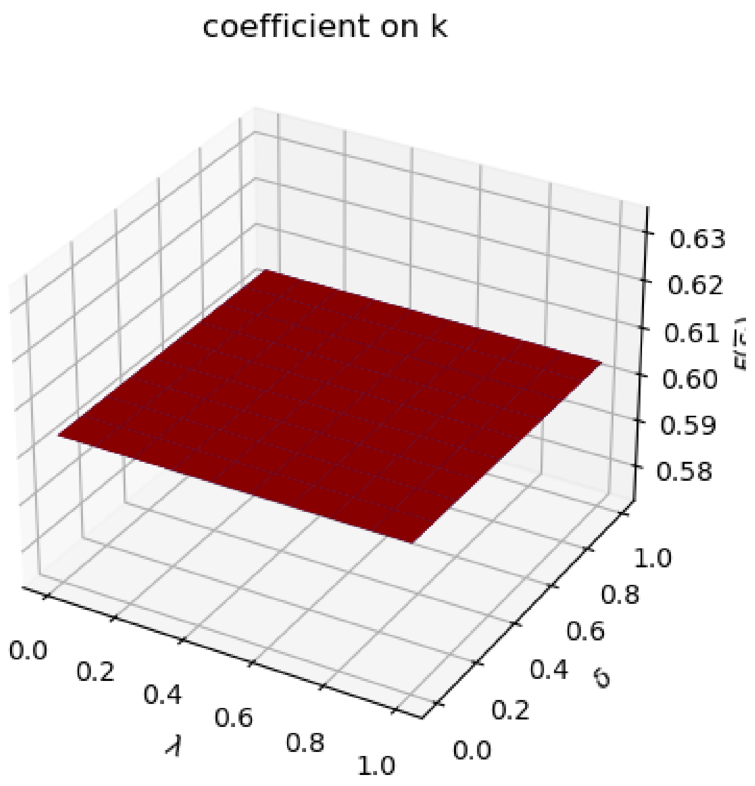
```
symmetric Π case:
```

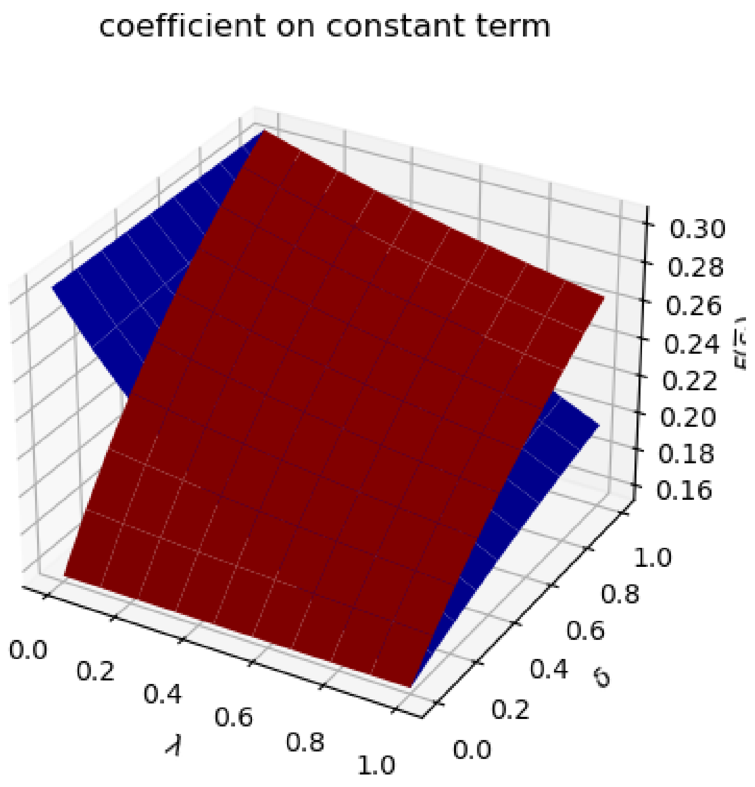

coefficient on k
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
0.63
0.62
0.61
0.60
0.59
0.58
0.0
0.2
0.4
0.6
0.8
1.0
λ

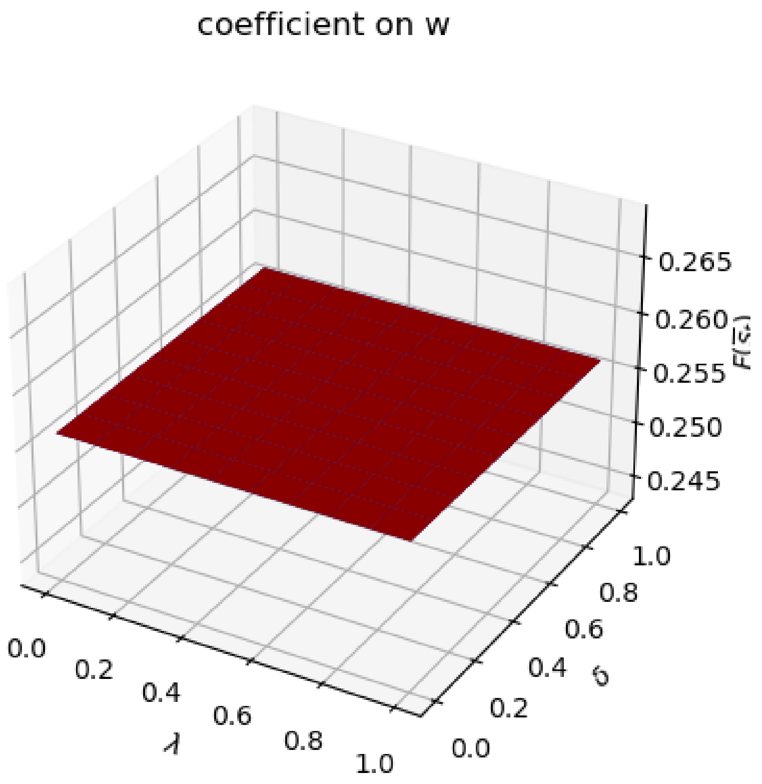

coefficient on constant term
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
λ
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0.0
0.2
0.4
0.6
0.8
1.0



```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
δ

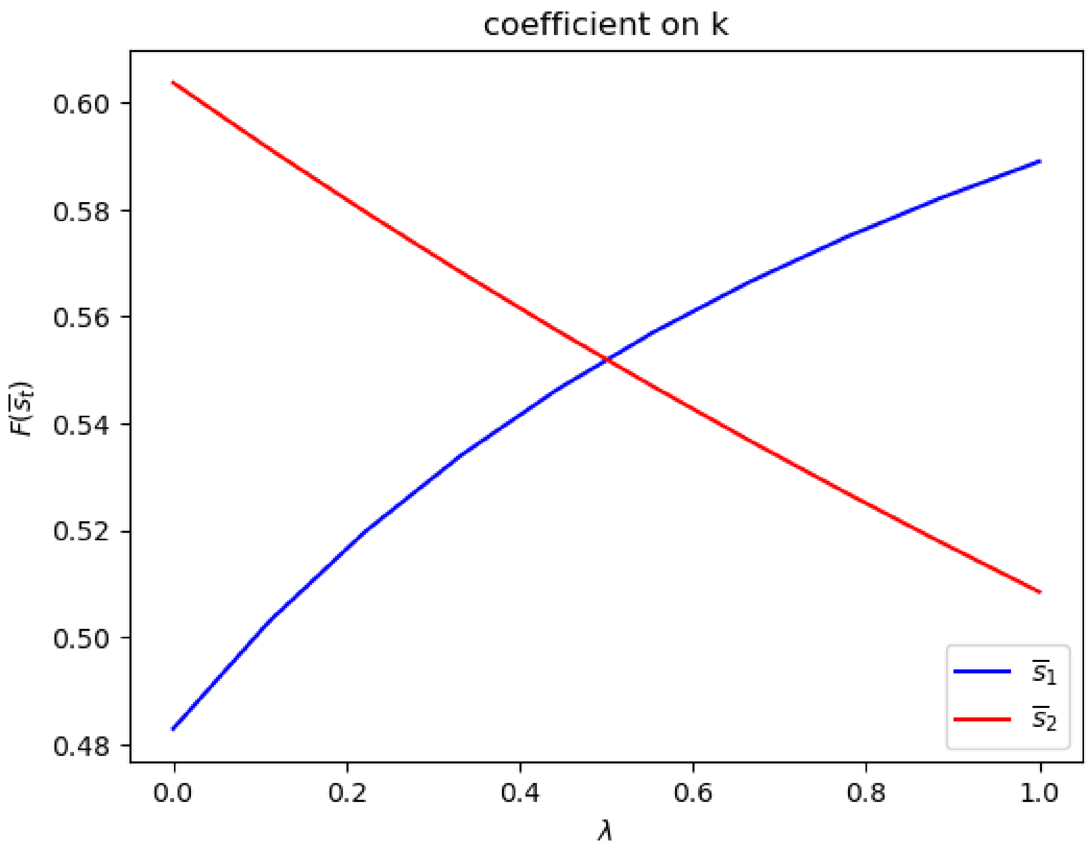

coefficient on constant term
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
δ

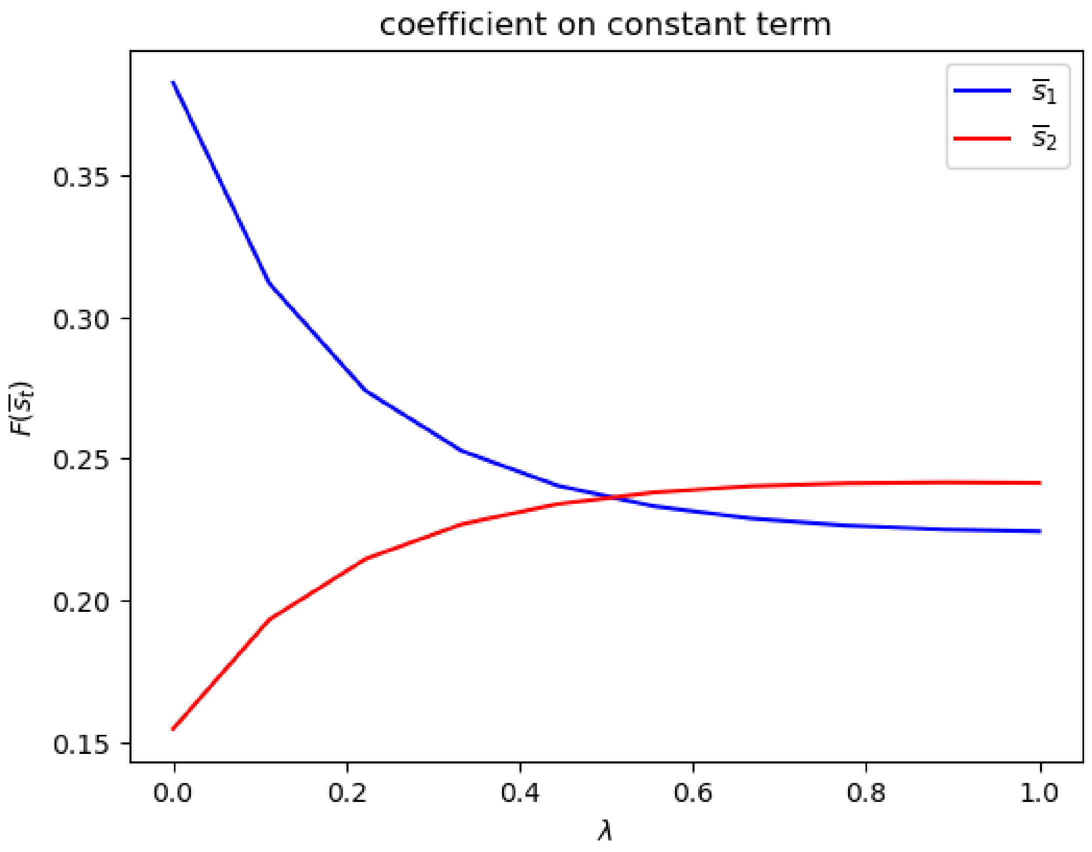


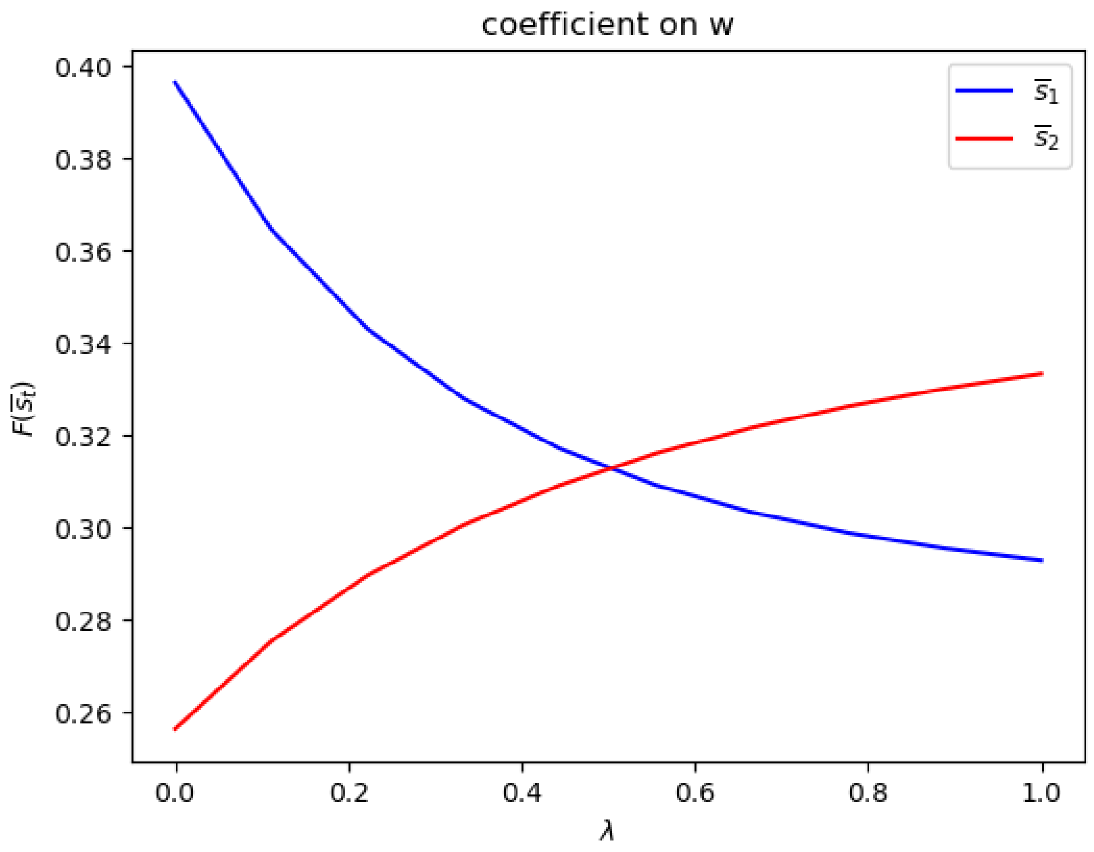
Only f_{2,s_t} depends on s_t.

```
run(construct_arrays2, {"f2_vals":[0.5, 1.]}, state_vec2)
```

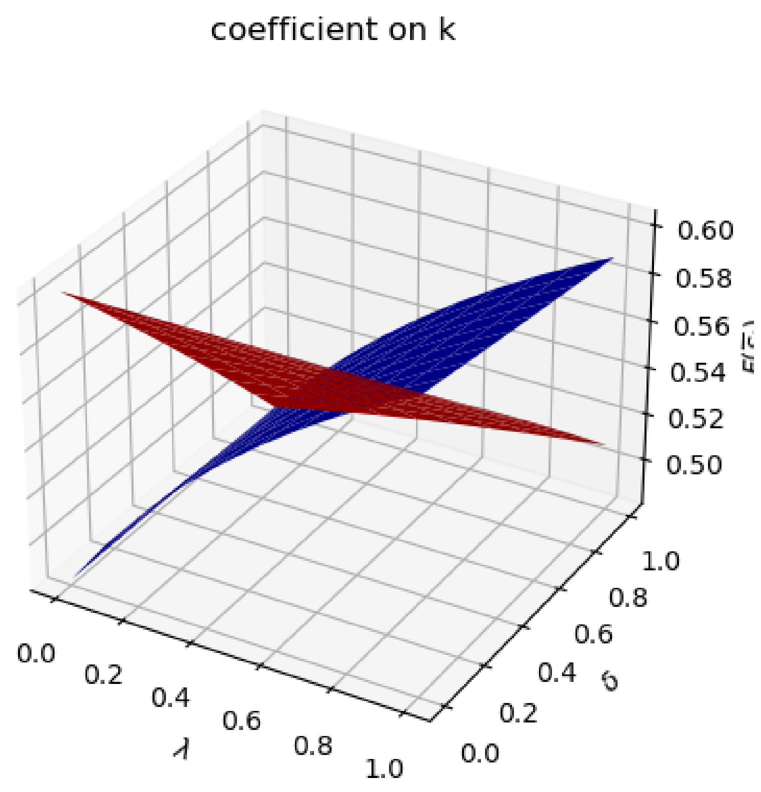
```
symmetric Π case:
```

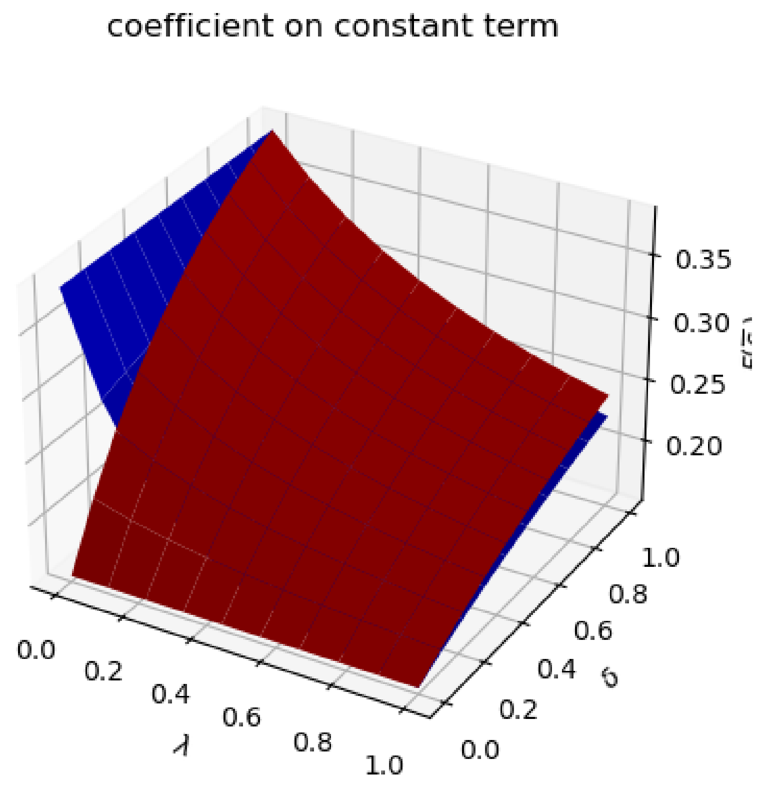

coefficient on k
$F(\overline{s}_t)$
λ
0.48
0.50
0.52
0.54
0.56
0.58
0.60
0.0
0.2
0.4
0.6
0.8
1.0
$\overline{s}_1$
$\overline{s}_2$

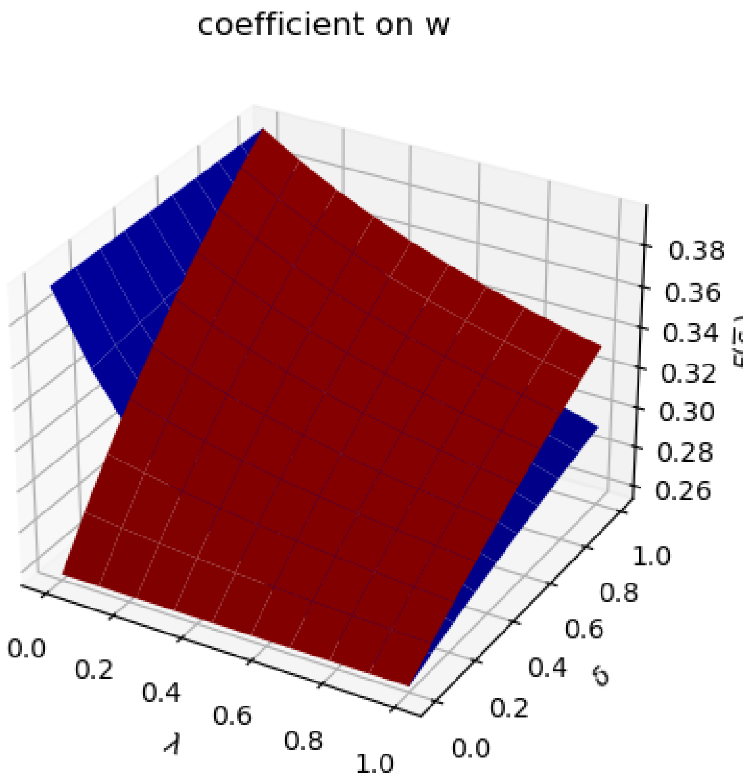

coefficient on constant term
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
0.35
0.30
0.25
0.20
0.15
0.0
0.2
0.4
0.6
0.8
1.0
λ



```
asymmetric Π case:
```


coefficient on k
0.60
0.58
0.56
0.54
0.52
0.50
0.0
0.2
0.4
0.6
0.8
1.0
λ
δ

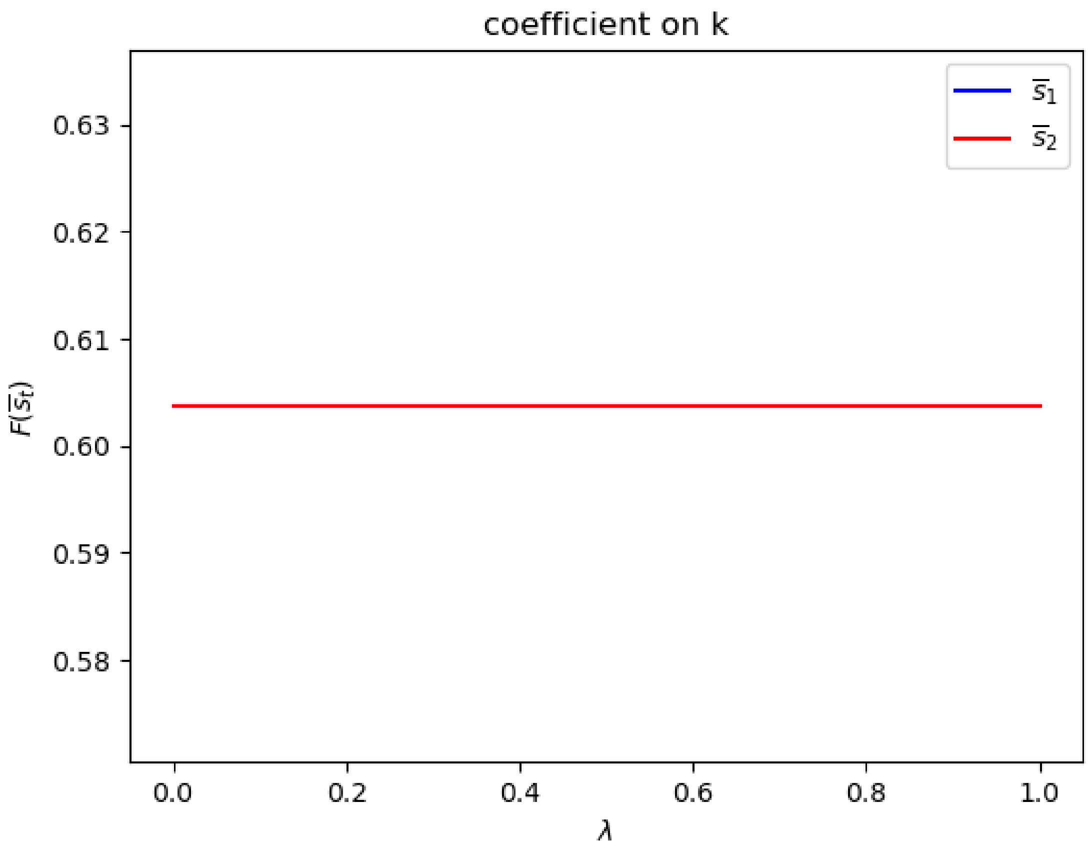

coefficient on constant term
0.35
0.30
0.25
0.20
0.0
0.2
0.4
0.6
0.8
1.0
λ
δ

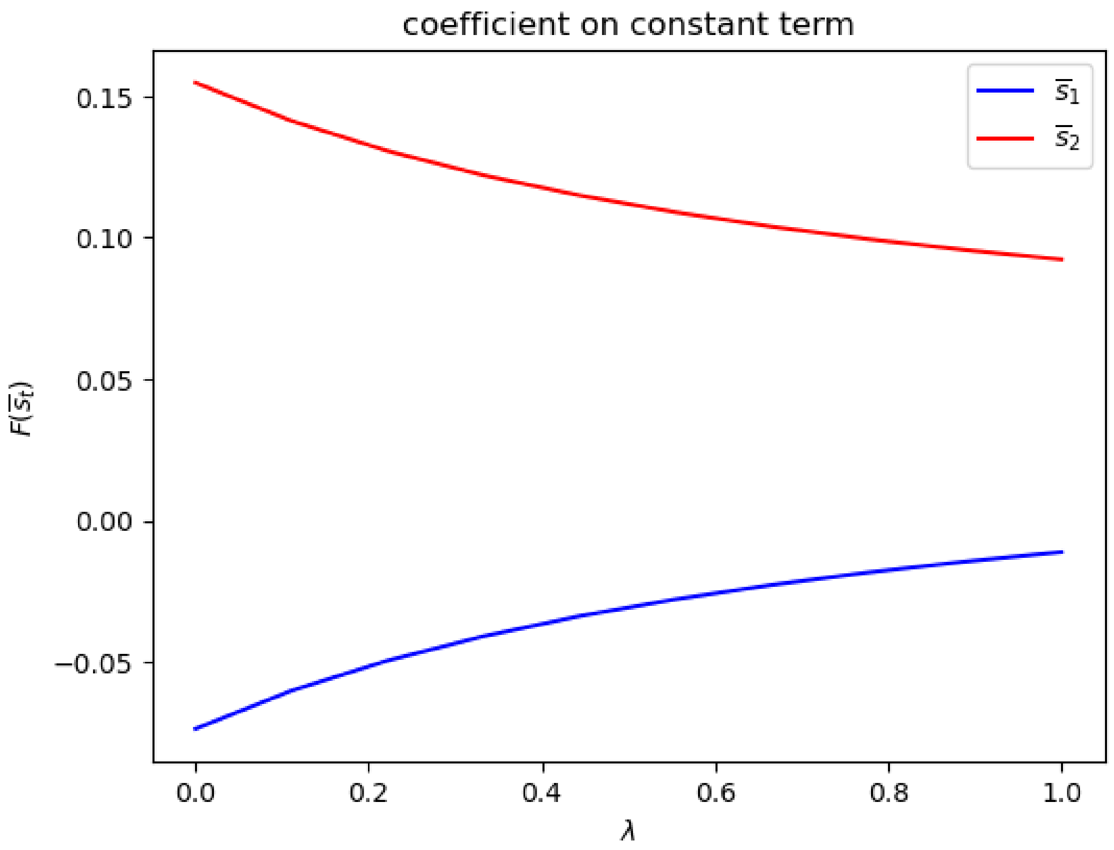


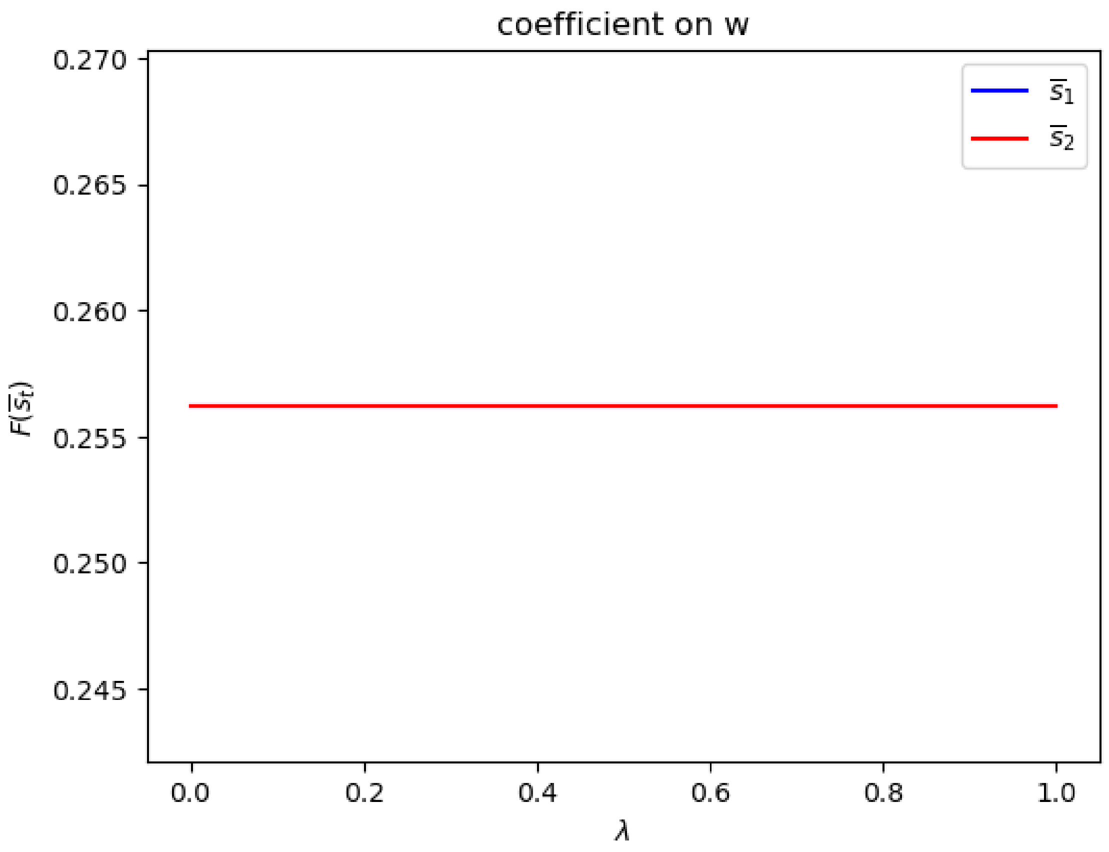
Only $\alpha_0(s_t)$ depends on s_t.

```
run(construct_arrays2, {"α0_vals":[0.5, 1.]}, state_vec2)
```

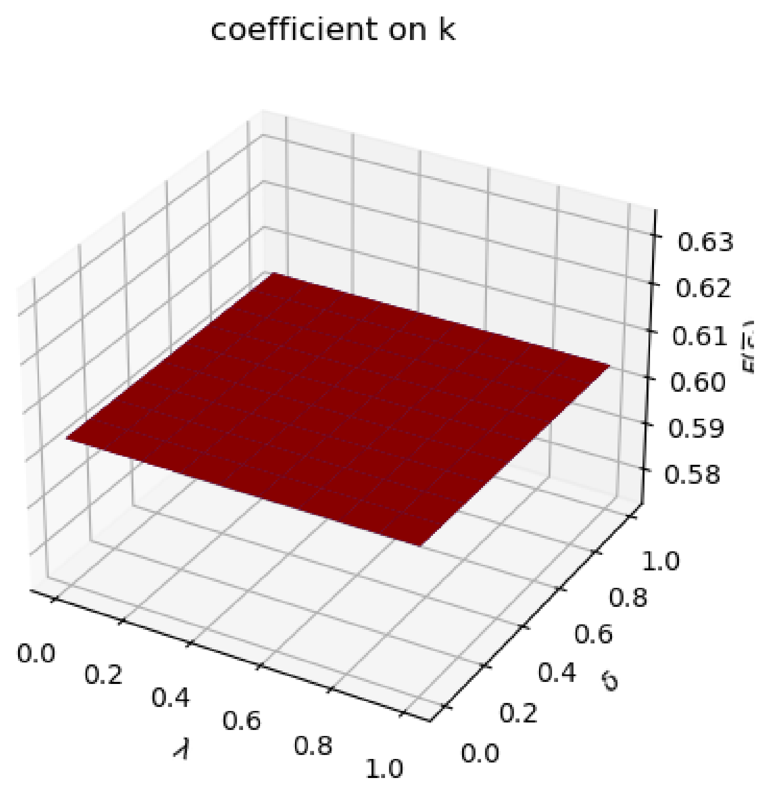
```
symmetric Π case:
```

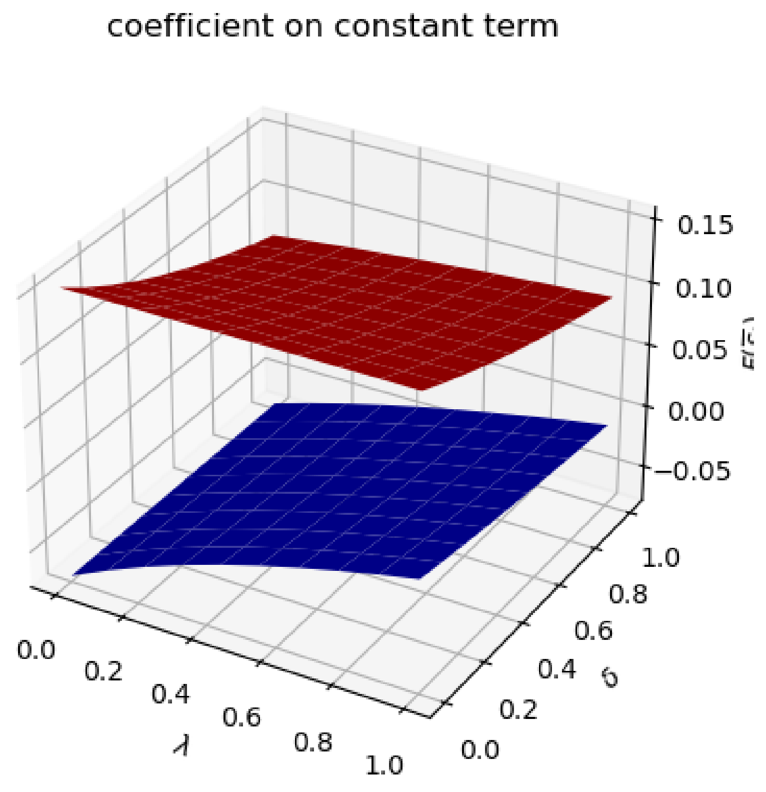

coefficient on k
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
λ
0.63
0.62
0.61
0.60
0.59
0.58
0.0
0.2
0.4
0.6
0.8
1.0

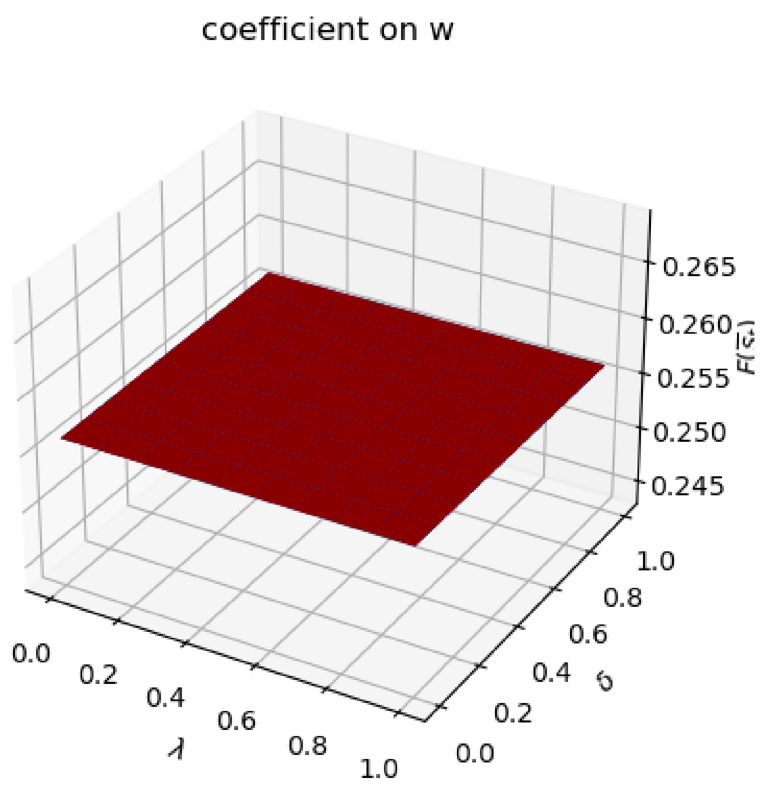

coefficient on constant term
$\overline{s}_1$
$\overline{s}_2$
0.15
0.10
0.05
0.00
−0.05
$F(\overline{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ



```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
F(ζ)
1.0
0.8
0.6
0.4
0.2
0.0
δ
0.0
0.2
0.4
0.6
0.8
1.0
λ

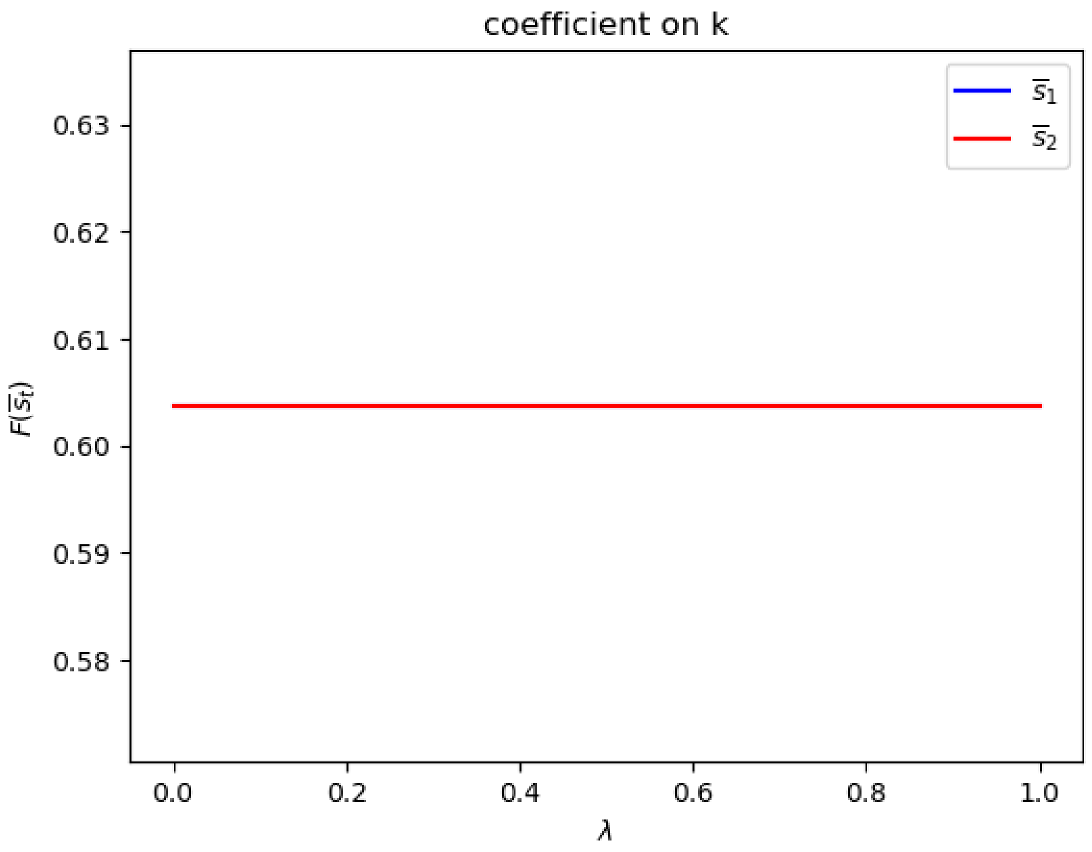

coefficient on constant term
0.15
0.10
0.05
0.00
−0.05
F(ζ)
1.0
0.8
0.6
0.4
0.2
0.0
δ
0.0
0.2
0.4
0.6
0.8
1.0
λ

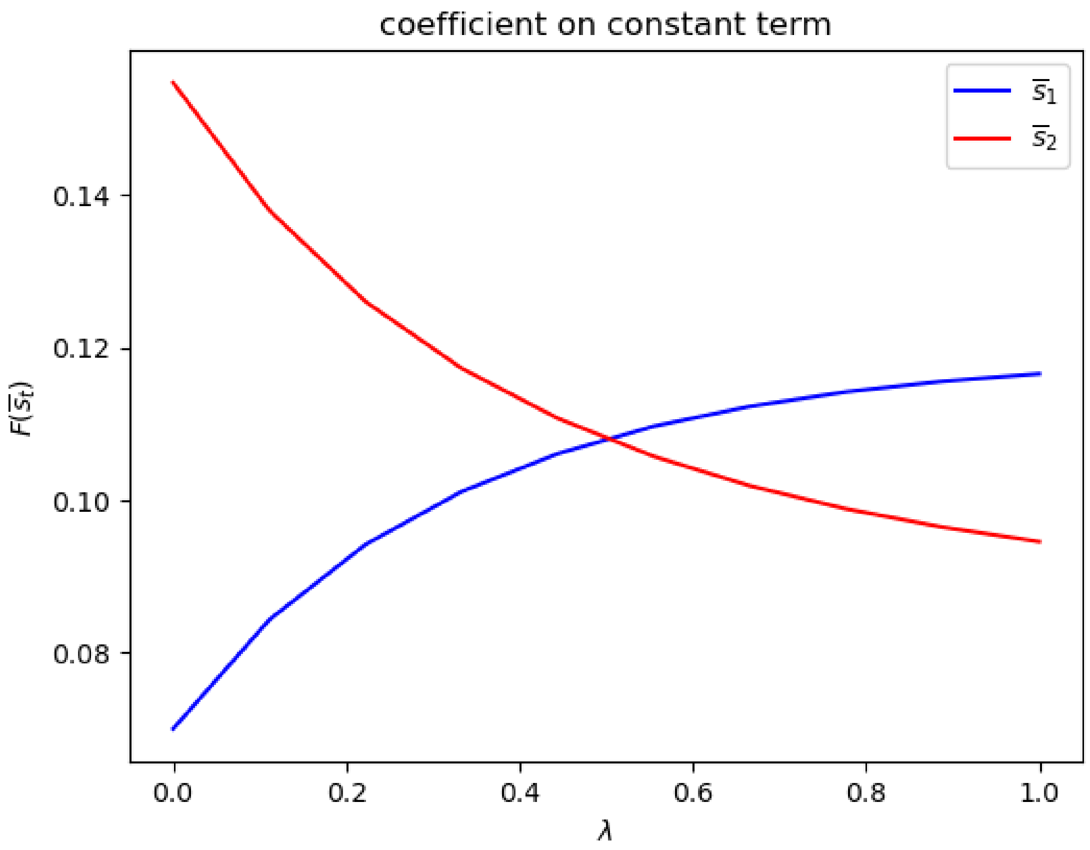


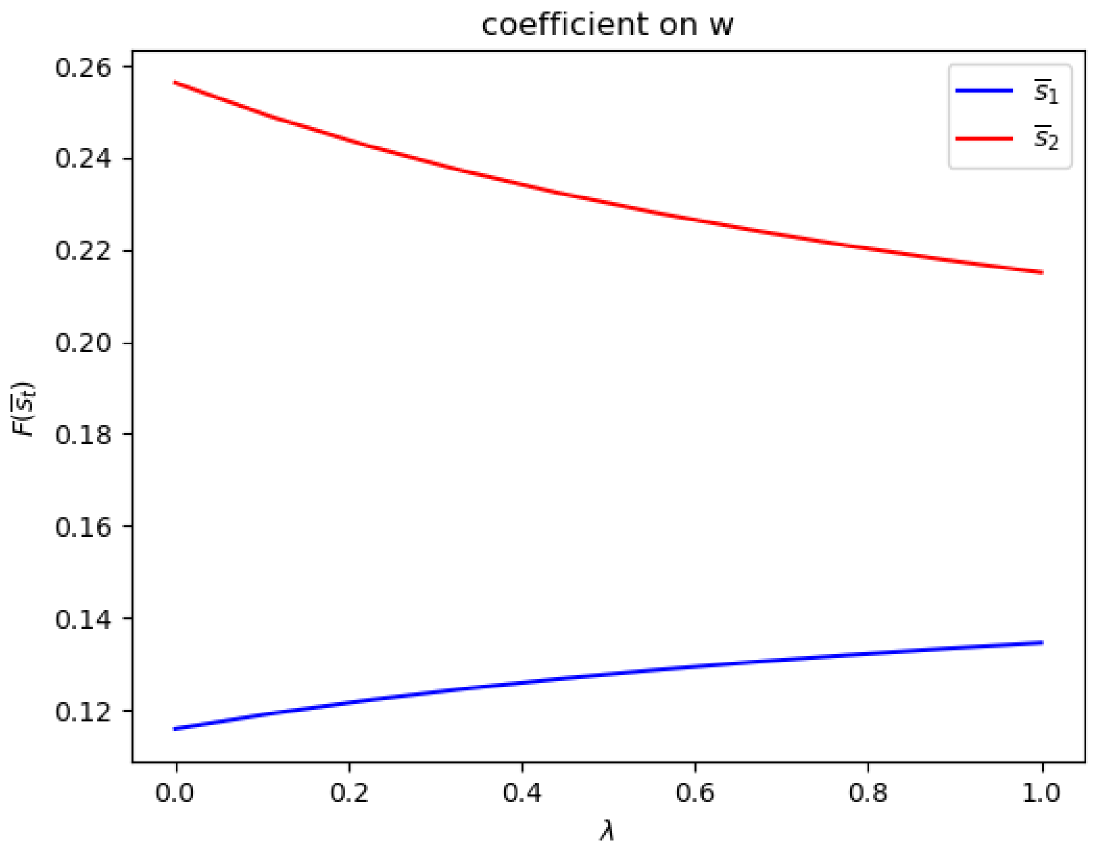
Only ρ_{s_t} depends on s_t.

```
run(construct_arrays2, {"ρ_vals":[0.5, 0.9]}, state_vec2)
```

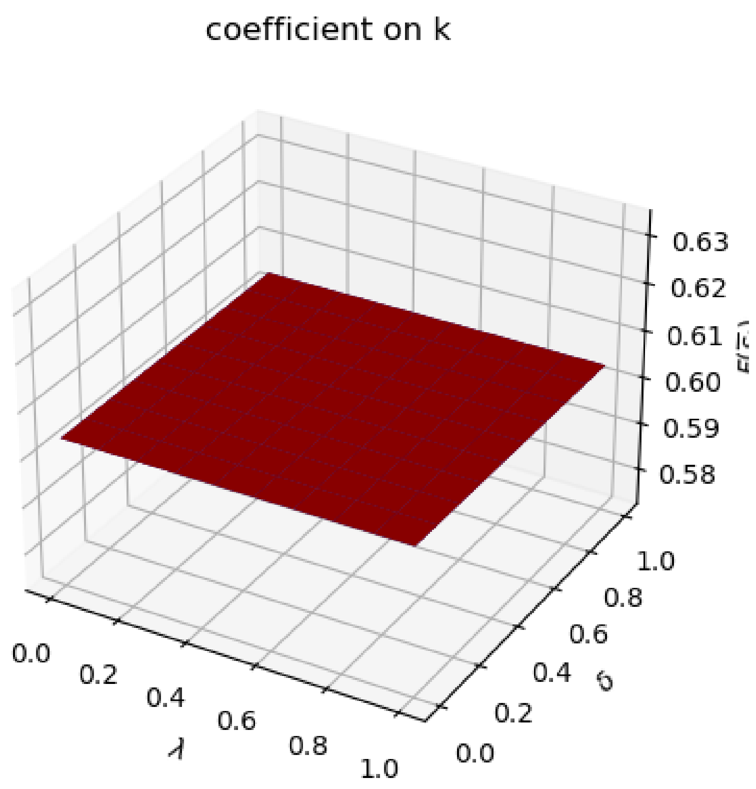
```
symmetric Π case:
```

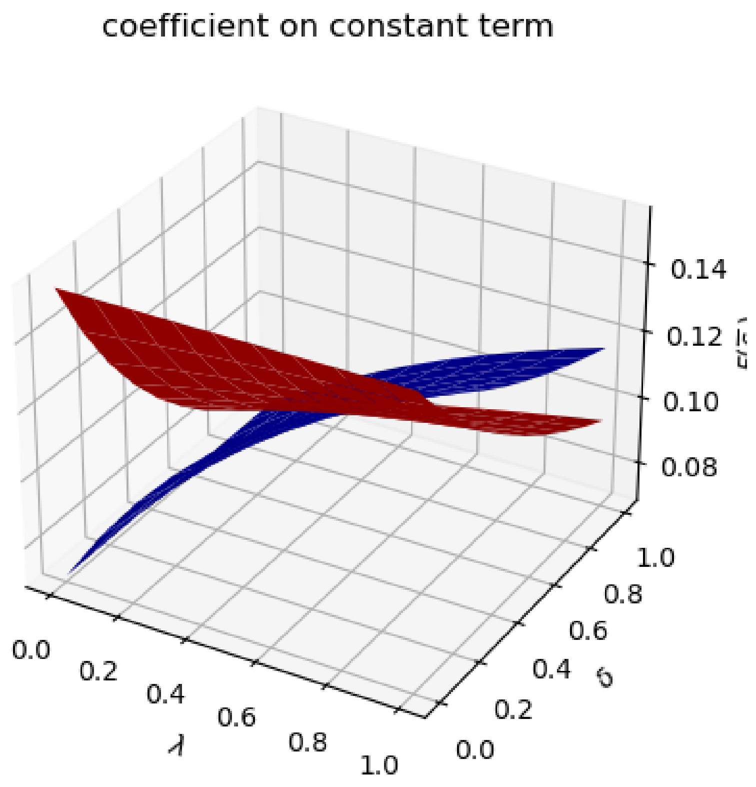

coefficient on k
$\overline{s}_1$
$\overline{s}_2$
0.63
0.62
0.61
0.60
0.59
0.58
$F(\overline{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ

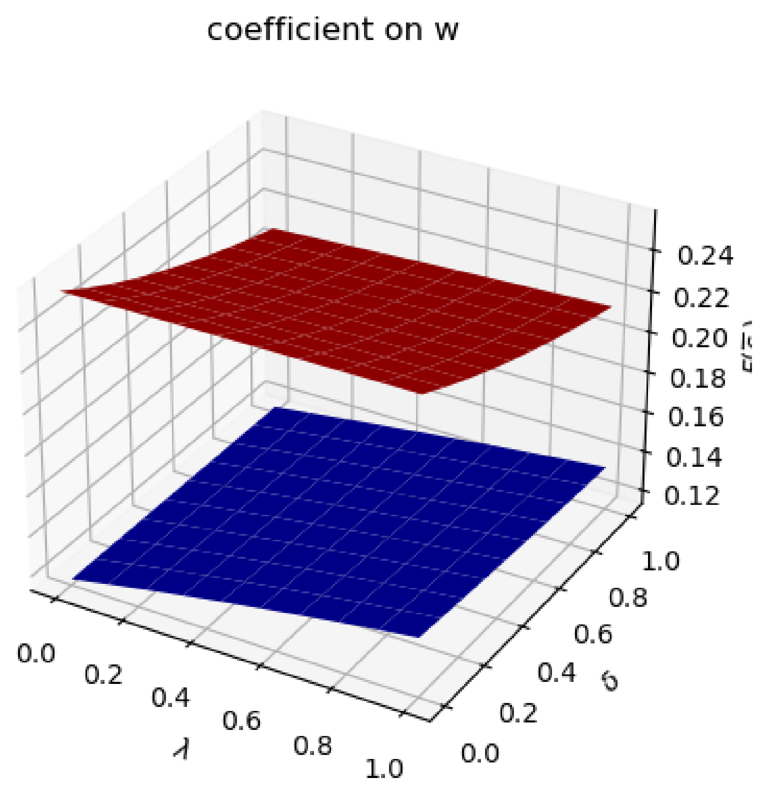

coefficient on constant term
$\overline{s}_1$
$\overline{s}_2$
$F(\overline{s}_t)$
0.14
0.12
0.10
0.08
0.0
0.2
0.4
0.6
0.8
1.0
λ



```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
0.0
0.2
0.4
0.6
0.8
1.0
λ
δ

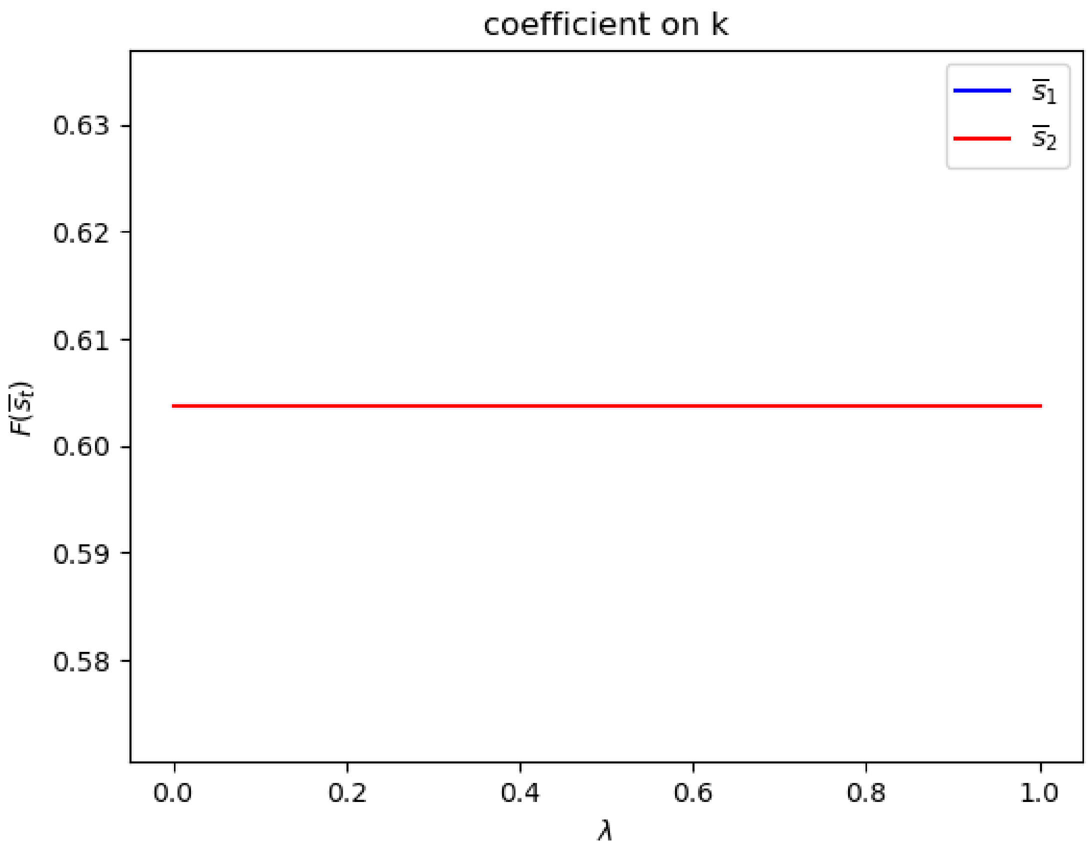

coefficient on constant term
0.14
0.12
0.10
0.08
0.0
0.2
0.4
0.6
0.8
1.0
λ
δ

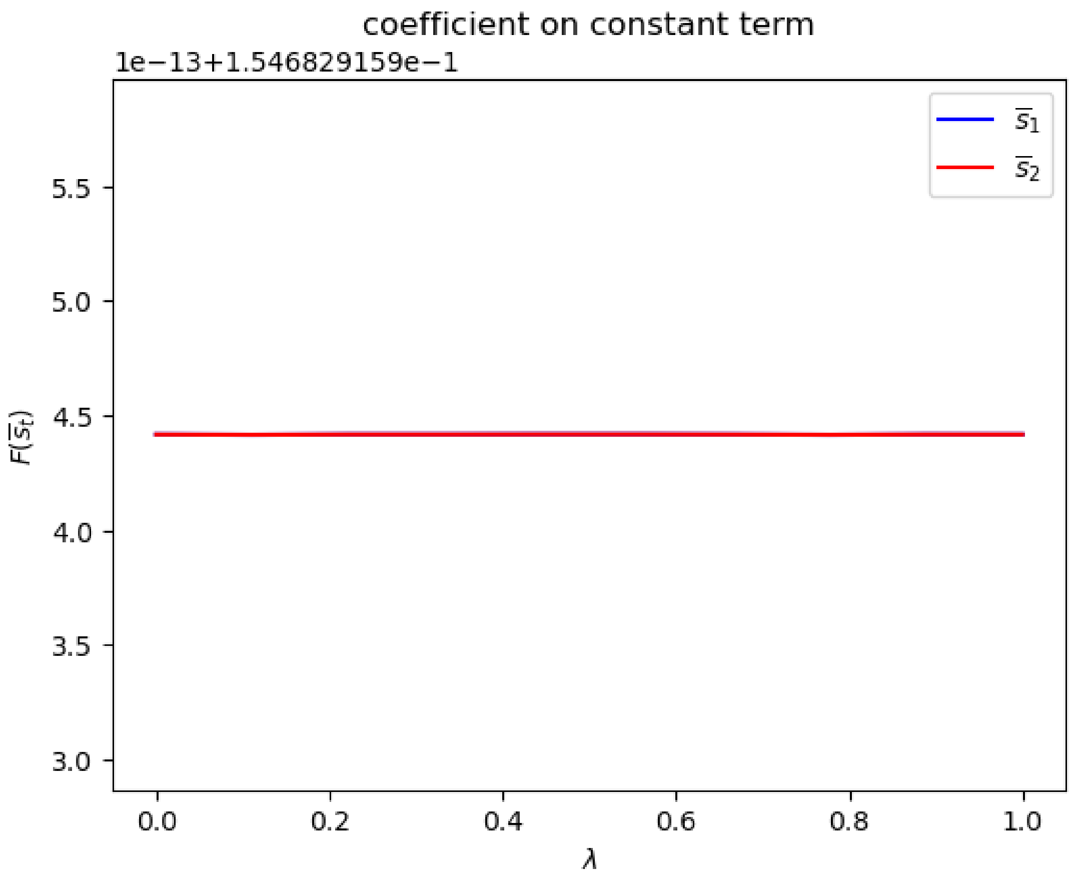


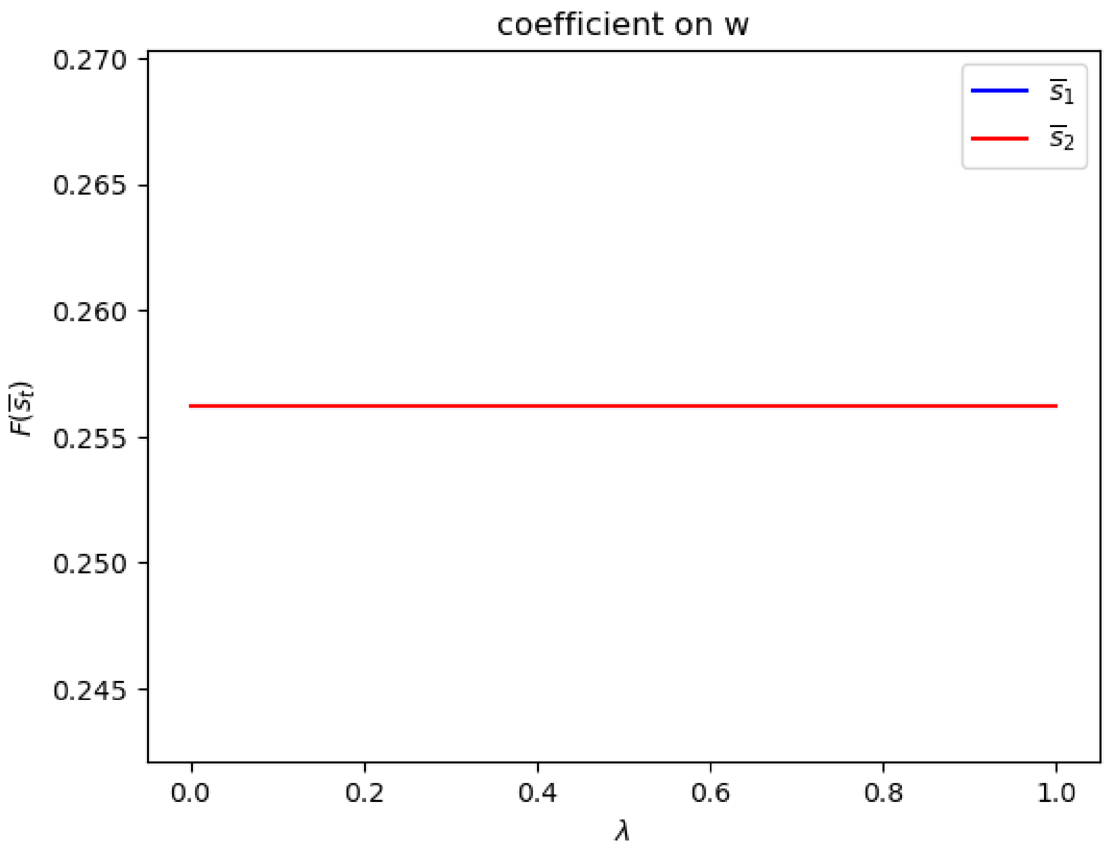
Only σ_{s_t} depends on s_t.

```
run(construct_arrays2, {"σ_vals":[0.5, 1.]}, state_vec2)
```

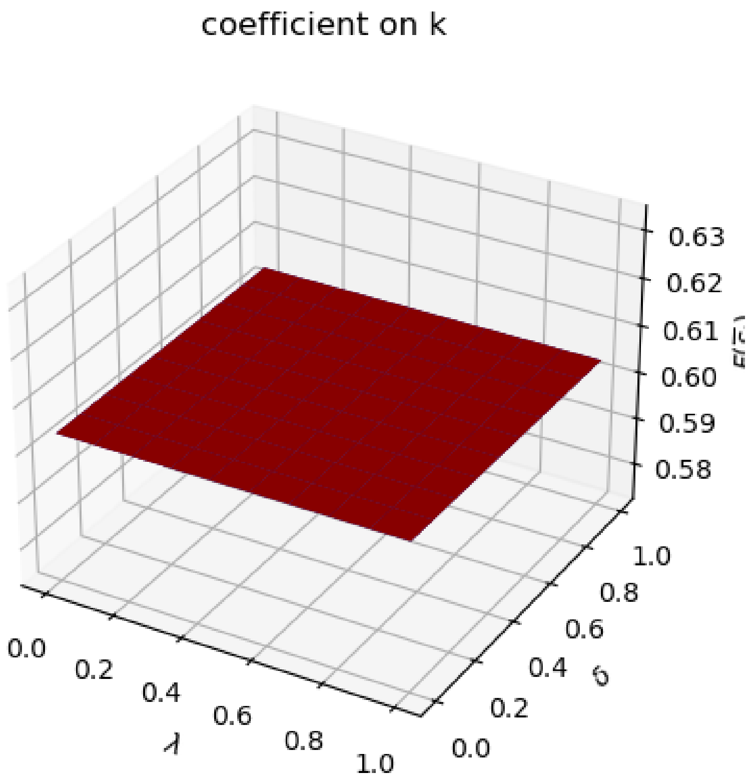
```
symmetric Π case:
```

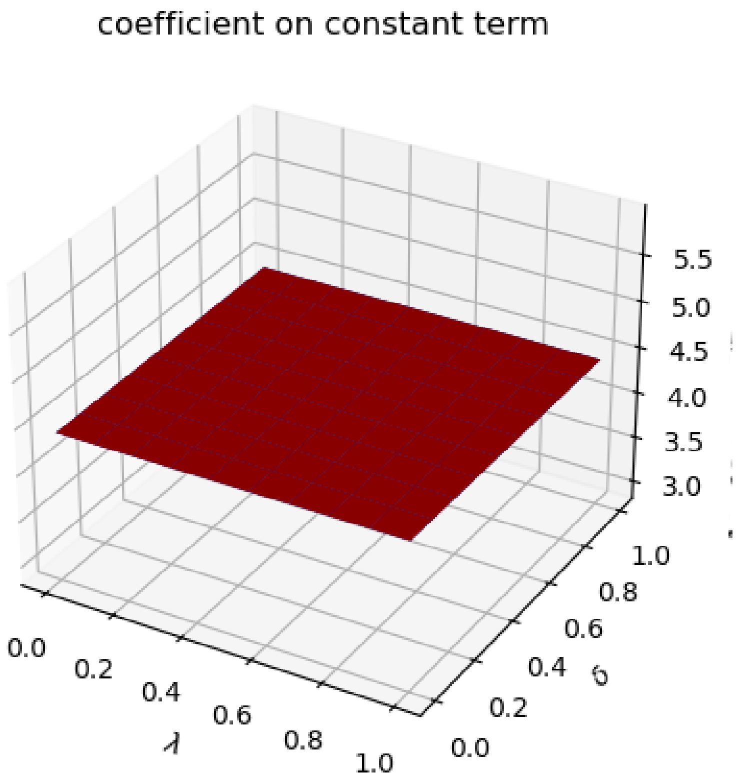

coefficient on k
$\overline{s}_1$
$\overline{s}_2$
0.63
0.62
0.61
0.60
0.59
0.58
$F(\overline{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ

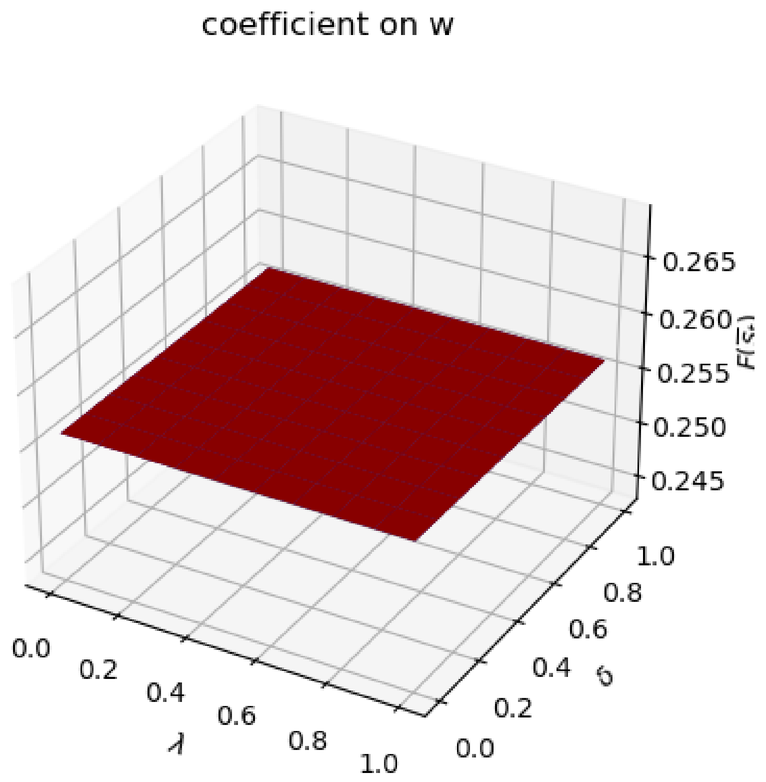

coefficient on constant term
1e−13+1.546829159e−1
$\overline{s}_1$
$\overline{s}_2$
5.5
5.0
4.5
4.0
3.5
3.0
$F(\overline{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ



```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
1.0
0.8
0.6
0.4
0.2
0.0
δ
0.0
0.2
0.4
0.6
0.8
1.0
λ


coefficient on constant term
5.5
5.0
4.5
4.0
3.5
3.0
1.0
0.8
0.6
0.4
0.2
0.0
δ
0.0
0.2
0.4
0.6
0.8
1.0
λ



13.7 More examples

The following lectures describe how Markov jump linear quadratic dynamic programming can be used to extend the [Barro, 1979] model of optimal tax-smoothing and government debt in several interesting directions

1. *How to Pay for a War: Part 1*
2. *How to Pay for a War: Part 2*
3. *How to Pay for a War: Part 3*

CHAPTER

FOURTEEN

HOW TO PAY FOR A WAR: PART 1

Contents

In addition to what's in Anaconda, this lecture will deploy quantecon:

```
!pip install --upgrade quantecon
```

14.1 Reader's Guide

Let's start with some standard imports:

```python
import quantecon as qe
import numpy as np
import matplotlib.pyplot as plt
```

This lecture uses the method of **Markov jump linear quadratic dynamic programming** that is described in lecture *Markov Jump LQ dynamic programming* to extend the [Barro, 1979] model of optimal tax-smoothing and government debt in a particular direction.

This lecture has two sequels that offer further extensions of the Barro model

1. *How to Pay for a War: Part 2*
2. *How to Pay for a War: Part 3*

The extensions are modified versions of his 1979 model later suggested by Barro (1999 [Barro, 1999], 2003 [Barro and McCleary, 2003]).

Barro's original 1979 [Barro, 1979] model is about a government that borrows and lends in order to minimize an intertemporal measure of distortions caused by taxes.

Technical tractability induced Barro [Barro, 1979] to assume that

- the government trades only one-period risk-free debt, and
- the one-period risk-free interest rate is constant

By using *Markov jump linear quadratic dynamic programming* we can allow interest rates to move over time in empirically interesting ways.

Also, by expanding the dimension of the state, we can add a maturity composition decision to the government's problem.

It is by doing these two things that we extend Barro's 1979 [Barro, 1979] model along lines he suggested in Barro (1999 [Barro, 1999], 2003 [Barro and McCleary, 2003]).

Barro (1979) [Barro, 1979] assumed

- that a government faces an **exogenous sequence** of expenditures that it must finance by a tax collection sequence whose expected present value equals the initial debt it owes plus the expected present value of those expenditures.
- that the government wants to minimize the following measure of tax distortions: $E_0 \sum_{t=0}^{\infty} \beta^t T_t^2$, where T_t are total tax collections and E_0 is a mathematical expectation conditioned on time 0 information.
- that the government trades only one asset, a risk-free one-period bond.
- that the gross interest rate on the one-period bond is constant and equal to β^{-1}, the reciprocal of the factor β at which the government discounts future tax distortions.

Barro's model can be mapped into a discounted linear quadratic dynamic programming problem.

Partly inspired by Barro (1999) [Barro, 1999] and Barro (2003) [Barro and McCleary, 2003], our generalizations of Barro's (1979) [Barro, 1979] model assume

- that the government borrows or saves in the form of risk-free bonds of maturities $1, 2, \ldots, H$.
- that interest rates on those bonds are time-varying and in particular, governed by a jointly stationary stochastic process.

Our generalizations are designed to fit within a generalization of an ordinary linear quadratic dynamic programming problem in which matrices that define the quadratic objective function and the state transition function are **time-varying** and **stochastic**.

This generalization, known as a **Markov jump linear quadratic dynamic program**, combines

- the computational simplicity of **linear quadratic dynamic programming**, and
- the ability of **finite state Markov chains** to represent interesting patterns of random variation.

We want the stochastic time variation in the matrices defining the dynamic programming problem to represent variation over time in

- interest rates
- default rates
- roll over risks

As described in *Markov Jump LQ dynamic programming*, the idea underlying **Markov jump linear quadratic dynamic programming** is to replace the constant matrices defining a **linear quadratic dynamic programming problem** with matrices that are fixed functions of an N state Markov chain.

For infinite horizon problems, this leads to N interrelated matrix Riccati equations that pin down N value functions and N linear decision rules, applying to the N Markov states.

14.2 Public Finance Questions

Barro's 1979 [Barro, 1979] model is designed to answer questions such as

- Should a government finance an exogenous surge in government expenditures by raising taxes or borrowing?
- How does the answer to that first question depend on the exogenous stochastic process for government expenditures, for example, on whether the surge in government expenditures can be expected to be temporary or permanent?

Barro's 1999 [Barro, 1999] and 2003 [Barro and McCleary, 2003] models are designed to answer more fine-grained questions such as

- What determines whether a government wants to issue short-term or long-term debt?
- How do roll-over risks affect that decision?
- How does the government's long-short *portfolio management* decision depend on features of the exogenous stochastic process for government expenditures?

Thus, both the simple and the more fine-grained versions of Barro's models are ways of precisely formulating the classic issue of *How to pay for a war*.

This lecture describes:

- An application of Markov jump LQ dynamic programming to a model in which a government faces exogenous time-varying interest rates for issuing one-period risk-free debt.

A *sequel to this lecture* describes applies Markov LQ control to settings in which a government issues risk-free debt of different maturities.

14.3 Barro (1979) Model

We begin by solving a version of the Barro (1979) [Barro, 1979] model by mapping it into the original LQ framework.

As mentioned in this lecture, the Barro model is mathematically isomorphic with the LQ permanent income model.

Let T_t denote tax collections, β a discount factor, $b_{t,t+1}$ time $t+1$ goods that the government promises to pay at t, G_t government purchases, $p_{t,t+1}$ the number of time t goods received per time $t+1$ goods promised.

Evidently, $p_{t,t+1}$ is inversely related to appropriate corresponding gross interest rates on government debt.

In the spirit of Barro (1979) [Barro, 1979], the stochastic process of government expenditures is exogenous.

The government's problem is to choose a plan for taxation and borrowing $\{b_{t+1}, T_t\}_{t=0}^{\infty}$ to minimize

$$E_0 \sum_{t=0}^{\infty} \beta^t T_t^2$$

subject to the constraints

$$T_t + p_{t,t+1} b_{t,t+1} = G_t + b_{t-1,t}$$

$$G_t = U_g z_t$$

$$z_{t+1} = A_{22} z_t + C_2 w_{t+1}$$

where $w_{t+1} \sim N(0, I)$

The variables $T_t, b_{t,t+1}$ are *control* variables chosen at t, while $b_{t-1,t}$ is an endogenous state variable inherited from the past at time t and $p_{t,t+1}$ is an exogenous state variable at time t.

To begin, we assume that $p_{t,t+1}$ is constant (and equal to β)

- later we will extend the model to allow $p_{t,t+1}$ to vary over time

To map into the LQ framework, we use $x_t = \begin{bmatrix} b_{t-1,t} \\ z_t \end{bmatrix}$ as the state vector, and $u_t = b_{t,t+1}$ as the control variable.

Therefore, the (A, B, C) matrices are defined by the state-transition law:

$$x_{t+1} = \begin{bmatrix} 0 & 0 \\ 0 & A_{22} \end{bmatrix} x_t + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_t + \begin{bmatrix} 0 \\ C_2 \end{bmatrix} w_{t+1}$$

To find the appropriate (R, Q, W) matrices, we note that G_t and $b_{t-1,t}$ can be written as appropriately defined functions of the current state:

$$G_t = S_G x_t \ \ , \ \ b_{t-1,t} = S_1 x_t$$

If we define $M_t = -p_{t,t+1}$, and let $S = S_G + S_1$, then we can write taxation as a function of the states and control using the government's budget constraint:

$$T_t = S x_t + M_t u_t$$

It follows that the (R, Q, W) matrices are implicitly defined by:

$$T_t^2 = x_t' S' S x_t + u_t' M_t' M_t u_t + 2 u_t' M_t' S x_t$$

If we assume that $p_{t,t+1} = \beta$, then $M_t \equiv M = -\beta$.

In this case, none of the LQ matrices are time varying, and we can use the original LQ framework.

We will implement this constant interest-rate version first, assuming that G_t follows an AR(1) process:

$$G_{t+1} = \bar{G} + \rho G_t + \sigma w_{t+1}$$

To do this, we set $z_t = \begin{bmatrix} 1 \\ G_t \end{bmatrix}$, and consequently:

$$A_{22} = \begin{bmatrix} 1 & 0 \\ \bar{G} & \rho \end{bmatrix} \ \ , \ \ C_2 = \begin{bmatrix} 0 \\ \sigma \end{bmatrix}$$

```
# Model parameters
β, Gbar, ρ, σ = 0.95, 5, 0.8, 1

# Basic model matrices
A22 = np.array([[1,    0],
                [Gbar, ρ],])

C2 = np.array([[0],
               [σ]])

Ug = np.array([[0, 1]])

# LQ framework matrices
A_t = np.zeros((1, 3))
A_b = np.hstack((np.zeros((2, 1)), A22))
A = np.vstack((A_t, A_b))

B = np.zeros((3, 1))
B[0, 0] = 1
```

(continues on next page)

(continued from previous page)

```
C = np.vstack((np.zeros((1, 1)), C2))

Sg = np.hstack((np.zeros((1, 1)), Ug))
S1 = np.zeros((1, 3))
S1[0, 0] = 1
S = S1 + Sg

M = np.array([[-β]])

R = S.T @ S
Q = M.T @ M
W = M.T @ S

# Small penalty on the debt required to implement the no-Ponzi scheme
R[0, 0] = R[0, 0] + 1e-9
```

We can now create an instance of `LQ`:

```
LQBarro = qe.LQ(Q, R, A, B, C=C, N=W, beta=β)
P, F, d = LQBarro.stationary_values()
x0 = np.array([[100, 1, 25]])
```

We can see the isomorphism by noting that consumption is a martingale in the permanent income model and that taxation is a martingale in Barro's model.

We can check this using the F matrix of the LQ model.

Because $u_t = -Fx_t$, we have

$$T_t = Sx_t + Mu_t = (S - MF)x_t$$

and

$$T_{t+1} = (S - MF)x_{t+1} = (S - MF)(Ax_t + Bu_t + Cw_{t+1}) = (S - MF)((A - BF)x_t + Cw_{t+1})$$

Therefore, the mathematical expectation of T_{t+1} conditional on time t information is

$$E_t T_{t+1} = (S - MF)(A - BF)x_t$$

Consequently, taxation is a martingale ($E_t T_{t+1} = T_t$) if

$$(S - MF)(A - BF) = (S - MF),$$

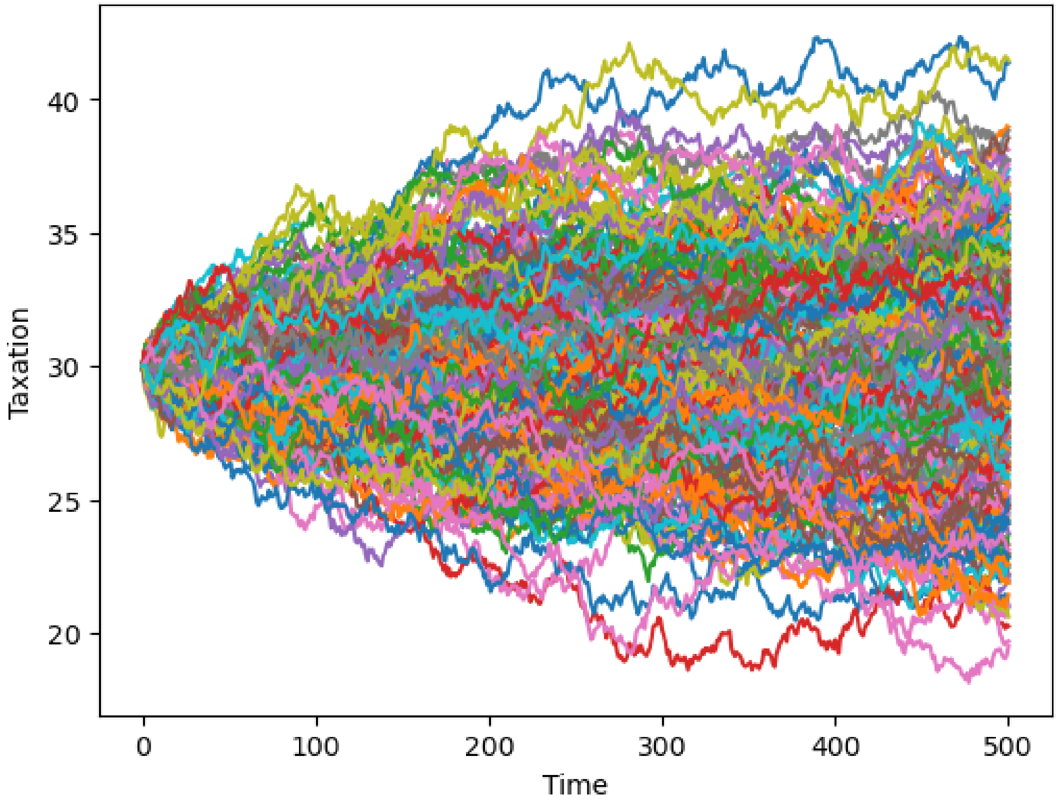
which holds in this case:

```
S - M @ F, (S - M @ F) @ (A - B @ F)
```

```
(array([[ 0.05000002, 19.79166502,  0.2083334 ]]),
 array([[ 0.05000002, 19.79166504,  0.2083334 ]]))
```

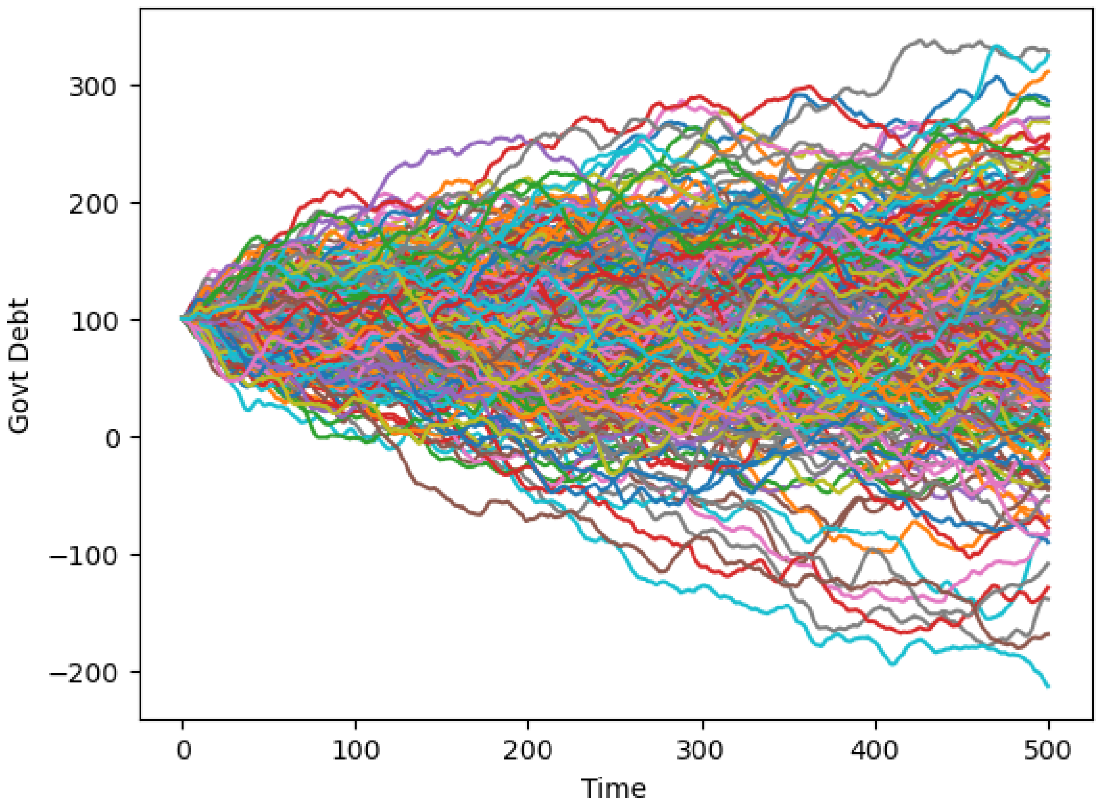
This explains the fanning out of the conditional empirical distribution of taxation across time, computing by simulation the Barro model a large number of times:

```
T = 500
for i in range(250):
    x, u, w = LQBarro.compute_sequence(x0, ts_length=T)
    plt.plot(list(range(T+1)), ((S - M @ F) @ x)[0, :])
plt.xlabel('Time')
plt.ylabel('Taxation')
plt.show()
```



We can see a similar, but a smoother pattern, if we plot government debt over time.

```
T = 500
for i in range(250):
    x, u, w = LQBarro.compute_sequence(x0, ts_length=T)
    plt.plot(list(range(T+1)), x[0, :])
plt.xlabel('Time')
plt.ylabel('Govt Debt')
plt.show()
```



14.4 Python Class to Solve Markov Jump Linear Quadratic Control Problems

To implement the extension to the Barro model in which $p_{t,t+1}$ varies over time, we must allow the M matrix to be time-varying.

Our Q and W matrices must also vary over time.

We can solve such a model using the `LQMarkov` class that solves Markov jump linear quandratic control problems as described above.

The code for the class can be viewed here.

The class takes lists of matrices that corresponds to N Markov states.

The value and policy functions are then found by iterating on a coupled system of matrix Riccati difference equations.

Optimal P_s, F_s, d_s are stored as attributes.

The class also contains a "method" for simulating the model.

14.5 Barro Model with a Time-varying Interest Rate

We can use the above class to implement a version of the Barro model with a time-varying interest rate. The simplest way to extend the model is to allow the interest rate to take two possible values. We set:

$$p^1_{t,t+1} = \beta + 0.02 = 0.97$$

$$p^2_{t,t+1} = \beta - 0.017 = 0.933$$

Thus, the first Markov state has a low interest rate, and the second Markov state has a high interest rate.

We also need to specify a transition matrix for the Markov state.

We use:

$$\Pi = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix}$$

(so each Markov state is persistent, and there is an equal chance of moving from one state to the other)

The choice of parameters means that the unconditional expectation of $p_{t,t+1}$ is 0.9515, higher than $\beta (= 0.95)$.

If we were to set $p_{t,t+1} = 0.9515$ in the version of the model with a constant interest rate, government debt would explode.

```python
# Create list of matrices that corresponds to each Markov state
Π = np.array([[0.8, 0.2],
              [0.2, 0.8]])

As = [A, A]
Bs = [B, B]
Cs = [C, C]
Rs = [R, R]

M1 = np.array([[-β - 0.02]])
M2 = np.array([[-β + 0.017]])

Q1 = M1.T @ M1
Q2 = M2.T @ M2
Qs = [Q1, Q2]
W1 = M1.T @ S
W2 = M2.T @ S
Ws = [W1, W2]

# create Markov Jump LQ DP problem instance
lqm = qe.LQMarkov(Π, Qs, Rs, As, Bs, Cs=Cs, Ns=Ws, beta=β)
lqm.stationary_values();
```

The decision rules are now dependent on the Markov state:

```python
lqm.Fs[0]
```

```
array([[-0.98437712, 19.20516427, -0.8314215 ]])
```

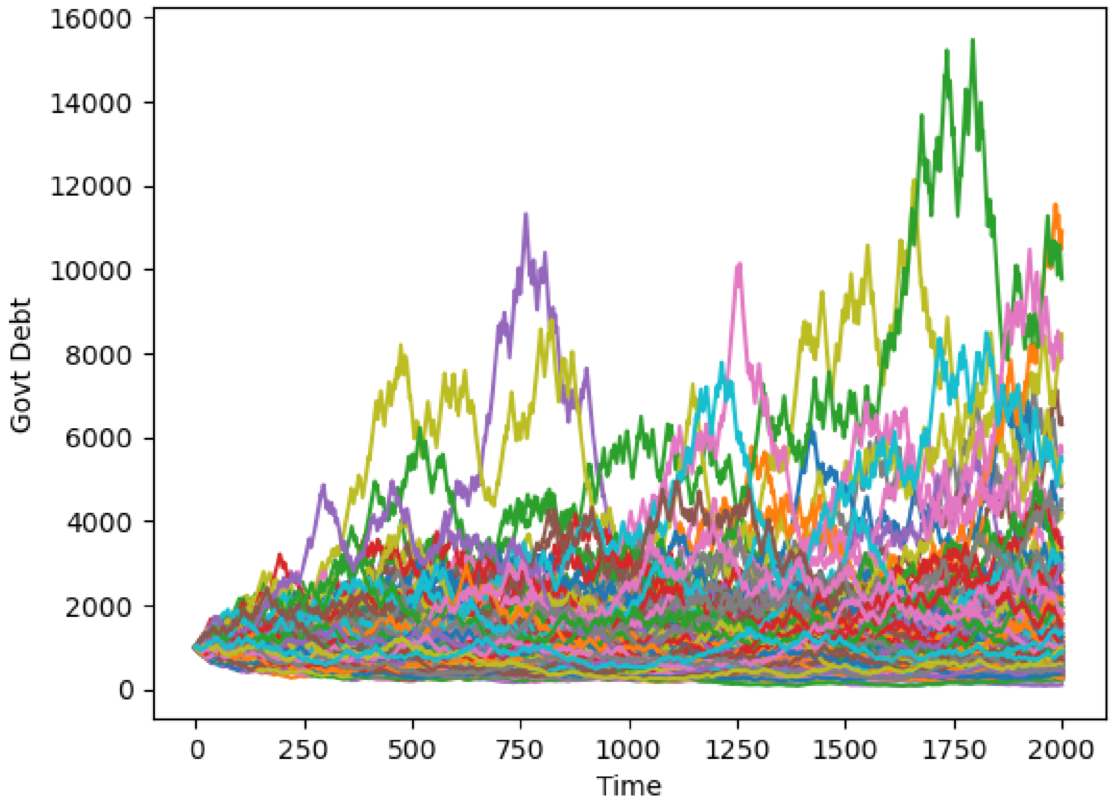
```python
lqm.Fs[1]
```

```
array([[-1.01434301, 21.5847983 , -0.83851116]])
```

Simulating a large number of such economies over time reveals interesting dynamics.

Debt tends to stay low and stable but recurrently surges.

```
T = 2000
x0 = np.array([[1000, 1, 25]])
for i in range(250):
    x, u, w, s = lqm.compute_sequence(x0, ts_length=T)
    plt.plot(list(range(T+1)), x[0, :])
plt.xlabel('Time')
plt.ylabel('Govt Debt')
plt.show()
```



CHAPTER

FIFTEEN

HOW TO PAY FOR A WAR: PART 2

Contents

In addition to what's in Anaconda, this lecture deploys the quantecon library:

```
!pip install --upgrade quantecon
```

15.1 An Application of Markov Jump Linear Quadratic Dynamic Programming

This is a *sequel to an earlier lecture*.

We use a method introduced in lecture *Markov Jump LQ dynamic programming* to implement suggestions by Barro (1999 [Barro, 1999], 2003 [Barro and McCleary, 2003]) for extending his classic 1979 model of tax smoothing.

Barro's 1979 [Barro, 1979] model is about a government that borrows and lends in order to help it minimize an intertemporal measure of distortions caused by taxes.

Technically, Barro's 1979 [Barro, 1979] model looks a lot like a consumption-smoothing model.

Our generalizations of his 1979 [Barro, 1979] model will also look like souped-up consumption-smoothing models.

Wanting tractability induced Barro in 1979 [Barro, 1979] to assume that

- the government trades only one-period risk-free debt, and
- the one-period risk-free interest rate is constant

In our *earlier lecture*, we relaxed the second of these assumptions but not the first.

In particular, we used *Markov jump linear quadratic dynamic programming* to allow the exogenous interest rate to vary over time.

In this lecture, we add a maturity composition decision to the government's problem by expanding the dimension of the state.

We assume

- that the government borrows or saves in the form of risk-free bonds of maturities $1, 2, \ldots, H$.
- that interest rates on those bonds are time-varying and in particular are governed by a jointly stationary stochastic process.

Let's start with some standard imports:

```
import quantecon as qe
import numpy as np
import matplotlib.pyplot as plt
```

15.2 Two example specifications

We'll describe two possible specifications

- In one, each period the government issues zero-coupon bonds of one- and two-period maturities and redeems them only when they mature – in this version, the maturity structure of government debt at each date is partly inherited from the past.
- In the second, the government redesigns the maturity structure of the debt each period.

15.3 One- and Two-period Bonds but No Restructuring

Let T_t denote tax collections, β a discount factor, $b_{t,t+1}$ time $t+1$ goods that the government promises to pay at t, $b_{t,t+2}$ time $t+2$ goods that the government promises to pay at time t, G_t government purchases, $p_{t,t+1}$ the number of time t goods received per time $t+1$ goods promised, and $p_{t,t+2}$ the number of time t goods received per time $t+2$ goods promised.

Evidently, $p_{t,t+1}, p_{t,t+2}$ are inversely related to appropriate corresponding gross interest rates on government debt.

In the spirit of Barro (1979) [Barro, 1979], government expenditures are governed by an exogenous stochastic process.

Given initial conditions $b_{-2,0}, b_{-1,0}, z_0, i_0$, where i_0 is the initial Markov state, the government chooses a contingency plan for $\{b_{t,t+1}, b_{t,t+2}, T_t\}_{t=0}^{\infty}$ to maximize.

$$-E_0 \sum_{t=0}^{\infty} \beta^t \left[T_t^2 + c_1 (b_{t,t+1} - b_{t,t+2})^2\right]$$

subject to the constraints

$$
\begin{aligned}
T_t &= G_t + b_{t-2,t} + b_{t-1,t} - p_{t,t+2} b_{t,t+2} - p_{t,t+1} b_{t,t+1} \\
G_t &= U_{g,s_t} z_t \\
z_{t+1} &= A_{22,s_t} z_t + C_{2,s_t} w_{t+1} \\
\begin{bmatrix} p_{t,t+1} \\ p_{t,t+2} \\ U_{g,s_t} \\ A_{22,s_t} \\ C_{2,s_t} \end{bmatrix} &\sim \text{functions of Markov state with transition matrix } \Pi
\end{aligned}
$$

Here $w_{t+1} \sim N(0, I)$ and Π_{ij} is the probability that the Markov state moves from state i to state j in one period.

The variables $T_t, b_{t,t+1}, b_{t,t+2}$ are *control* variables chosen at t, while the variables $b_{t-1,t}, b_{t-2,t}$ are endogenous state variables inherited from the past at time t and $p_{t,t+1}, p_{t,t+2}$ are exogenous state variables at time t.

The parameter c_1 imposes a penalty on the government's issuing different quantities of one and two-period debt.

This penalty deters the government from taking large "long-short" positions in debt of different maturities. An example below will show this in action.

As well as extending the model to allow for a maturity decision for government debt, we can also in principle allow the matrices $U_{g,s_t}, A_{22,s_t}, C_{2,s_t}$ to depend on the Markov state s_t.

Below, we will often adopt the convention that for matrices appearing in a linear state space, $A_t \equiv A_{s_t}, C_t \equiv C_{s_t}$ and so on, so that dependence on t is always intermediated through the Markov state s_t.

15.4 Mapping into an LQ Markov Jump Problem

First, define

$$\hat{b}_t = b_{t-1,t} + b_{t-2,t},$$

which is debt due at time t.

Then define the endogenous part of the state:

$$\bar{b}_t = \begin{bmatrix} \hat{b}_t \\ b_{t-1,t+1} \end{bmatrix}$$

and the complete state

$$x_t = \begin{bmatrix} \bar{b}_t \\ z_t \end{bmatrix}$$

and the control vector

$$u_t = \begin{bmatrix} b_{t,t+1} \\ b_{t,t+2} \end{bmatrix}$$

The endogenous part of state vector follows the law of motion:

$$\begin{bmatrix} \hat{b}_{t+1} \\ b_{t,t+2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{b}_t \\ b_{t-1,t+1} \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b_{t,t+1} \\ b_{t,t+2} \end{bmatrix}$$

or

$$\bar{b}_{t+1} = A_{11} \bar{b}_t + B_1 u_t$$

Define the following functions of the state

$$G_t = S_{G,t} x_t, \quad \hat{b}_t = S_1 x_t$$

and

$$M_t = \begin{bmatrix} -p_{t,t+1} & -p_{t,t+2} \end{bmatrix}$$

where $p_{t,t+1}$ is the discount on one period loans in the discrete Markov state at time t and $p_{t,t+2}$ is the discount on two-period loans in the discrete Markov state.

Define

$$S_t = S_{G,t} + S_1$$

Note that in discrete Markov state i

$$T_t = M_t u_t + S_t x_t$$

It follows that

$$T_t^2 = x_t' S_t' S_t x_t + u_t' M_t' M_t u_t + 2 u_t' M_t' S_t x_t$$

or

$$T_t^2 = x_t' R_t x_t + u_t' Q_t u_t + 2 u_t' W_t x_t$$

where

$$R_t = S_t' S_t, \quad Q_t = M_t' M_t, \quad W_t = M_t' S_t$$

Because the payoff function also includes the penalty parameter on issuing debt of different maturities, we have:

$$T_t^2 + c_1 (b_{t,t+1} - b_{t,t+2})^2 = x_t' R_t x_t + u_t' Q_t u_t + 2 u_t' W_t x_t + c_1 u_t' Q^c u_t$$

where $Q^c = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. Therefore, the overall Q matrix for the Markov jump LQ problem is:

$$Q_t^c = Q_t + c_1 Q^c$$

The law of motion of the state in all discrete Markov states i is

$$x_{t+1} = A_t x_t + B u_t + C_t w_{t+1}$$

where

$$A_t = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22,t} \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ 0 \end{bmatrix}, \quad C_t = \begin{bmatrix} 0 \\ C_{2,t} \end{bmatrix}$$

Thus, in this problem all the matrices apart from B may depend on the Markov state at time t.

As shown in the *previous lecture*, the `LQMarkov` class can solve Markov jump LQ problems when provided with the A, B, C, R, Q, W matrices for each Markov state.

The function below maps the primitive matrices and parameters from the above two-period model into the matrices that the `LQMarkov` class requires:

```python
def LQ_markov_mapping(A22, C2, Ug, p1, p2, c1=0):

    """
    Function which takes A22, C2, Ug, p_{t, t+1}, p_{t, t+2} and penalty
    parameter c1, and returns the required matrices for the LQMarkov
    model: A, B, C, R, Q, W.
    This version uses the condensed version of the endogenous state.
    """

    # Make sure all matrices can be treated as 2D arrays
    A22 = np.atleast_2d(A22)
    C2 = np.atleast_2d(C2)
    Ug = np.atleast_2d(Ug)
    p1 = np.atleast_2d(p1)
    p2 = np.atleast_2d(p2)

    # Find the number of states (z) and shocks (w)
    nz, nw = C2.shape

    # Create A11, B1, S1, S2, Sg, S matrices
    A11 = np.zeros((2, 2))
    A11[0, 1] = 1

    B1 = np.eye(2)

    S1 = np.hstack((np.eye(1), np.zeros((1, nz+1))))
    Sg = np.hstack((np.zeros((1, 2)), Ug))
    S = S1 + Sg

    # Create M matrix
    M = np.hstack((-p1, -p2))

    # Create A, B, C matrices
    A_T = np.hstack((A11, np.zeros((2, nz))))
    A_B = np.hstack((np.zeros((nz, 2)), A22))
    A = np.vstack((A_T, A_B))

    B = np.vstack((B1, np.zeros((nz, 2))))

    C = np.vstack((np.zeros((2, nw)), C2))

    # Create Q^c matrix
    Qc = np.array([[1, -1], [-1, 1]])

    # Create R, Q, W matrices

    R = S.T @ S
    Q = M.T @ M + c1 * Qc
    W = M.T @ S

    return A, B, C, R, Q, W
```

With the above function, we can proceed to solve the model in two steps:

1. Use `LQ_markov_mapping` to map $U_{g,t}, A_{22,t}, C_{2,t}, p_{t,t+1}, p_{t,t+2}$ into the A, B, C, R, Q, W matrices for each of the n Markov states.
2. Use the `LQMarkov` class to solve the resulting n-state Markov jump LQ problem.

15.5 Penalty on Different Issuance Across Maturities

To implement a simple example of the two-period model, we assume that G_t follows an AR(1) process:

$$G_{t+1} = \bar{G} + \rho G_t + \sigma w_{t+1}$$

To do this, we set $z_t = \begin{bmatrix} 1 \\ G_t \end{bmatrix}$, and consequently:

$$A_{22} = \begin{bmatrix} 1 & 0 \\ \bar{G} & \rho \end{bmatrix} \quad , \quad C_2 = \begin{bmatrix} 0 \\ \sigma \end{bmatrix} \quad , \quad U_g = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

Therefore, in this example, A_{22}, C_2 and U_g are not time-varying.

We will assume that there are two Markov states, one with a flatter yield curve, and one with a steeper yield curve. In state 1, prices are:

$$p^1_{t,t+1} = \beta \quad , \quad p^1_{t,t+2} = \beta^2 - 0.02$$

and in state 2, prices are:

$$p^2_{t,t+1} = \beta \quad , \quad p^2_{t,t+2} = \beta^2 + 0.02$$

We first solve the model with no penalty parameter on different issuance across maturities, i.e. $c_1 = 0$.

We also need to specify a transition matrix for the Markov state, we use:

$$\Pi = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$$

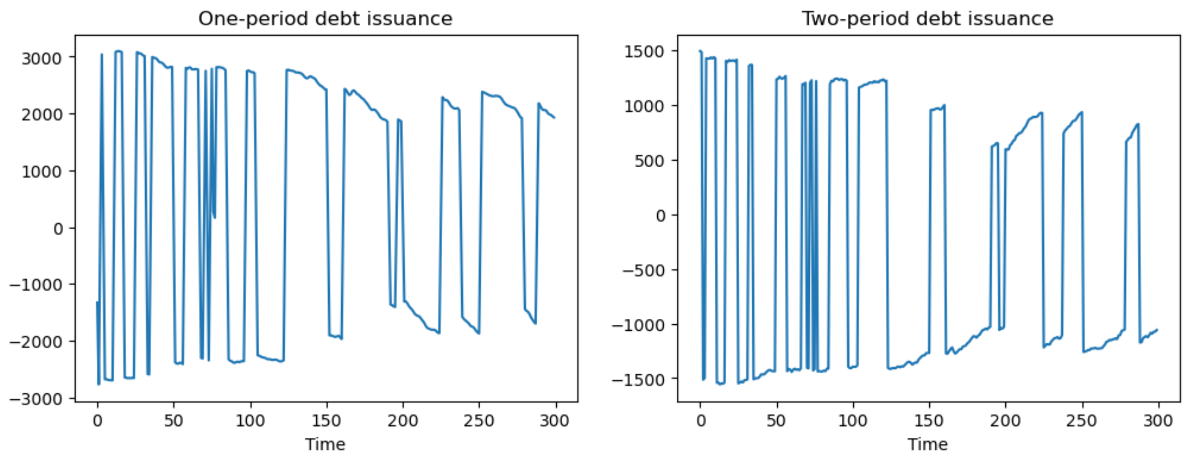
Thus, each Markov state is persistent, and there is an equal chance of moving from one to the other.

```
# Model parameters
β, Gbar, ρ, σ, c1 = 0.95, 5, 0.8, 1, 0
p1, p2, p3, p4 = β, β**2 - 0.02, β, β**2 + 0.02

# Basic model matrices
A22 = np.array([[1, 0], [Gbar, ρ] ,])
C_2 = np.array([[0], [σ]])
Ug = np.array([[0, 1]])

A1, B1, C1, R1, Q1, W1 = LQ_markov_mapping(A22, C_2, Ug, p1, p2, c1)
A2, B2, C2, R2, Q2, W2 = LQ_markov_mapping(A22, C_2, Ug, p3, p4, c1)

# Small penalties on debt required to implement no-Ponzi scheme
R1[0, 0] = R1[0, 0] + 1e-9
R2[0, 0] = R2[0, 0] + 1e-9

# Construct lists of matrices correspond to each state
As = [A1, A2]
Bs = [B1, B2]
Cs = [C1, C2]
Rs = [R1, R2]
Qs = [Q1, Q2]
Ws = [W1, W2]

Π = np.array([[0.9, 0.1],
```

(continues on next page)

(continued from previous page)

```
               [0.1, 0.9]])

# Construct and solve the model using the LQMarkov class
lqm = qe.LQMarkov(Π, Qs, Rs, As, Bs, Cs=Cs, Ns=Ws, beta=β)
lqm.stationary_values()

# Simulate the model
x0 = np.array([[100, 50, 1, 10]])
x, u, w, t = lqm.compute_sequence(x0, ts_length=300)

# Plot of one and two-period debt issuance
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(u[0, :])
ax1.set_title('One-period debt issuance')
ax1.set_xlabel('Time')
ax2.plot(u[1, :])
ax2.set_title('Two-period debt issuance')
ax2.set_xlabel('Time')
plt.show()
```



The above simulations show that when no penalty is imposed on different issuances across maturities, the government has an incentive to take large "long-short" positions in debt of different maturities.

To prevent such an outcome, we now set $c_1 = 0.01$.

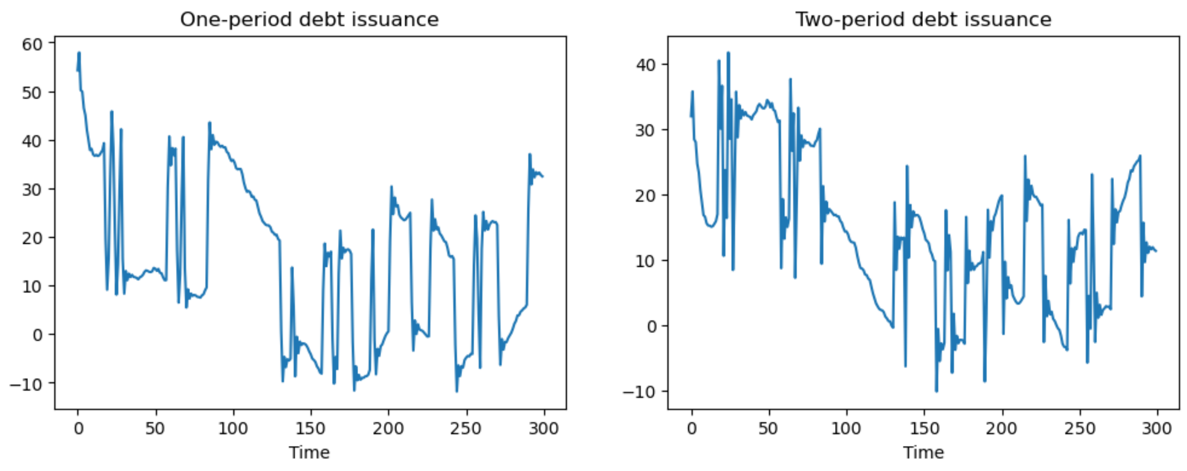
This penalty is enough to ensure that the government issues positive quantities of both one and two-period debt:

```
# Put small penalty on different issuance across maturities
c1 = 0.01

A1, B1, C1, R1, Q1, W1 = LQ_markov_mapping(A22, C_2, Ug, p1, p2, c1)
A2, B2, C2, R2, Q2, W2 = LQ_markov_mapping(A22, C_2, Ug, p3, p4, c1)

# Small penalties on debt required to implement no-Ponzi scheme
R1[0, 0] = R1[0, 0] + 1e-9
R2[0, 0] = R2[0, 0] + 1e-9

# Construct lists of matrices
As = [A1, A2]
Bs = [B1, B2]
```

(continues on next page)

(continued from previous page)

```
Cs = [C1, C2]
Rs = [R1, R2]
Qs = [Q1, Q2]
Ws = [W1, W2]

# Construct and solve the model using the LQMarkov class
lqm2 = qe.LQMarkov(Π, Qs, Rs, As, Bs, Cs=Cs, Ns=Ws, beta=β)
lqm2.stationary_values()

# Simulate the model
x, u, w, t = lqm2.compute_sequence(x0, ts_length=300)

# Plot of one and two-period debt issuance
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(u[0, :])
ax1.set_title('One-period debt issuance')
ax1.set_xlabel('Time')
ax2.plot(u[1, :])
ax2.set_title('Two-period debt issuance')
ax2.set_xlabel('Time')
plt.show()
```



15.6 A Model with Restructuring

This model alters two features of the previous model:

1. The maximum horizon of government debt is now extended to a general H periods.
2. The government is able to redesign the maturity structure of debt every period.

We impose a cost on adjusting issuance of each maturity by amending the payoff function to become:

$$T_t^2 + \sum_{j=0}^{H-1} c_2 (b_{t+j}^{t-1} - b_{t+j+1}^{t})^2$$

The government's budget constraint is now:

$$T_t + \sum_{j=1}^{H} p_{t,t+j} b_{t+j}^{t} = b_t^{t-1} + \sum_{j=1}^{H-1} p_{t,t+j} b_{t+j}^{t-1} + G_t$$

To map this into the Markov Jump LQ framework, we define state and control variables.

Let:

$$\bar{b}_t = \begin{bmatrix} b_t^{t-1} \\ b_{t+1}^{t-1} \\ \vdots \\ b_{t+H-1}^{t-1} \end{bmatrix} \quad , \quad u_t = \begin{bmatrix} b_{t+1}^t \\ b_{t+2}^t \\ \vdots \\ b_{t+H}^t \end{bmatrix}$$

Thus, $\bar{b}_t$ is the endogenous state (debt issued last period) and u_t is the control (debt issued today).

As before, we will also have the exogenous state z_t, which determines government spending.

Therefore, the full state is:

$$x_t = \begin{bmatrix} \bar{b}_t \\ z_t \end{bmatrix}$$

We also define a vector p_t that contains the time t price of goods in period $t+j$:

$$p_t = \begin{bmatrix} p_{t,t+1} \\ p_{t,t+2} \\ \vdots \\ p_{t,t+H} \end{bmatrix}$$

Finally, we define three useful matrices $S_s, S_x, \tilde{S}_x$:

$$\begin{bmatrix} p_{t,t+1} \\ p_{t,t+2} \\ \vdots \\ p_{t,t+H-1} \end{bmatrix} = S_s p_t \text{ where } S_s = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} b_{t+1}^{t-1} \\ b_{t+2}^{t-1} \\ \vdots \\ b_{t+T-1}^{t-1} \end{bmatrix} = S_x \bar{b}_t \text{ where } S_x = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

$$b_t^{t-1} = \tilde{S}_x \bar{b}_t \text{ where } \tilde{S}_x = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

In terms of dimensions, the first two matrices defined above are $(H-1) \times H$.

The last is $1 \times H$

We can now write the government's budget constraint in matrix notation. Rearranging the government budget constraint gives:

$$T_t = b_t^{t-1} + \sum_{j=1}^{H-1} p_{t+j}^t b_{t+j}^{t-1} + G_t - \sum_{j=1}^{H} p_{t+j}^t b_{t+j}^t$$

or

$$T_t = \tilde{S}_x \bar{b}_t + (S_s p_t) \cdot (S_x \bar{b}_t) + U_g z_t - p_t \cdot u_t$$

If we want to write this in terms of the full state, we have:

$$T_t = \begin{bmatrix} (\tilde{S}_x + p_t' S_s' S_x) & Ug \end{bmatrix} x_t - p_t' u_t$$

To simplify the notation, let $S_t = \begin{bmatrix} (\tilde{S}_x + p_t ' S_s ' S_x) & Ug \end{bmatrix}$.

Then

$$T_t = S_t x_t - p_t' u_t$$

Therefore

$$T_t^2 = x_t' R_t x_t + u_t' Q_t u_t + 2 u_t' W_t x_t$$

where

$$R_t = S_t' S_t, \quad Q_t = p_t p_t', \quad W_t = -p_t S_t$$

where to economize on notation we adopt the convention that for the linear state matrices $R_t \equiv R_{s_t}, Q_t \equiv W_{s_t}$ and so on.

We'll continue to use this convention also for the linear state matrices A, B, W and so on below.

Because the payoff function also includes the penalty parameter for rescheduling, we have:

$$T_t^2 + \sum_{j=0}^{H-1} c_2 (b_{t+j}^{t-1} - b_{t+j+1}^t)^2 = T_t^2 + c_2 (\bar{b}_t - u_t)' (\bar{b}_t - u_t)$$

Because the complete state is x_t and not $\bar{b}_t$, we rewrite this as:

$$T_t^2 + c_2 (S_c x_t - u_t)' (S_c x_t - u_t)$$

where $S_c = \begin{bmatrix} I & 0 \end{bmatrix}$

Multiplying this out gives:

$$T_t^2 + c_2 x_t' S_c' S_c x_t - 2 c_2 u_t' S_c x_t + c_2 u_t' u_t$$

Therefore, with the cost term, we must amend our R, Q, W matrices as follows:

$$R_t^c = R_t + c_2 S_c' S_c$$

$$Q_t^c = Q_t + c_2 I$$

$$W_t^c = W_t - c_2 S_c$$

To finish mapping into the Markov jump LQ setup, we need to construct the law of motion for the full state.

This is simpler than in the previous setup, as we now have $\bar{b}_{t+1} = u_t$.

Therefore:

$$x_{t+1} \equiv \begin{bmatrix} \bar{b}_{t+1} \\ z_{t+1} \end{bmatrix} = A_t x_t + B u_t + C_t w_{t+1}$$

where

$$A_t = \begin{bmatrix} 0 & 0 \\ 0 & A_{22,t} \end{bmatrix}, \quad B = \begin{bmatrix} I \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 \\ C_{2,t} \end{bmatrix}$$

This completes the mapping into a Markov jump LQ problem.

15.7 Restructuring as a Markov Jump Linear Quadratic Control Problem

As with the previous model, we can use a function to map the primitives of the model with restructuring into the matrices that the `LQMarkov` class requires:

```python
def LQ_markov_mapping_restruct(A22, C2, Ug, T, p_t, c=0):

    """
    Function which takes A22, C2, T, p_t, c and returns the
    required matrices for the LQMarkov model: A, B, C, R, Q, W
    Note, p_t should be a T by 1 matrix
    c is the rescheduling cost (a scalar)
    This version uses the condensed version of the endogenous state
    """

    # Make sure all matrices can be treated as 2D arrays
    A22 = np.atleast_2d(A22)
    C2 = np.atleast_2d(C2)
    Ug = np.atleast_2d(Ug)
    p_t = np.atleast_2d(p_t)

    # Find the number of states (z) and shocks (w)
    nz, nw = C2.shape

    # Create Sx, tSx, Ss, S_t matrices (tSx stands for \tilde S_x)
    Ss = np.hstack((np.eye(T-1), np.zeros((T-1, 1))))
    Sx = np.hstack((np.zeros((T-1, 1)), np.eye(T-1)))
    tSx = np.zeros((1, T))
    tSx[0, 0] = 1

    S_t = np.hstack((tSx + p_t.T @ Ss.T @ Sx, Ug))

    # Create A, B, C matrices
    A_T = np.hstack((np.zeros((T, T)), np.zeros((T, nz))))
    A_B = np.hstack((np.zeros((nz, T)), A22))
    A = np.vstack((A_T, A_B))

    B = np.vstack((np.eye(T), np.zeros((nz, T))))
    C = np.vstack((np.zeros((T, nw)), C2))

    # Create cost matrix Sc
    Sc = np.hstack((np.eye(T), np.zeros((T, nz))))

    # Create R_t, Q_t, W_t matrices

    R_c = S_t.T @ S_t + c * Sc.T @ Sc
    Q_c = p_t @ p_t.T + c * np.eye(T)
    W_c = -p_t @ S_t - c * Sc

    return A, B, C, R_c, Q_c, W_c
```

15.7.1 Example with Restructuring

As an example of the model with restructuring, consider this model where $H = 3$.

We will assume that there are two Markov states, one with a flatter yield curve, and one with a steeper yield curve.

In state 1, prices are:

$$p^1_{t,t+1} = 0.9695 \quad , \quad p^1_{t,t+2} = 0.902 \quad , \quad p^1_{t,t+3} = 0.8369$$

and in state 2, prices are:

$$p^2_{t,t+1} = 0.9295 \quad , \quad p^2_{t,t+2} = 0.902 \quad , \quad p^2_{t,t+3} = 0.8769$$

We will assume the same transition matrix and G_t process as above

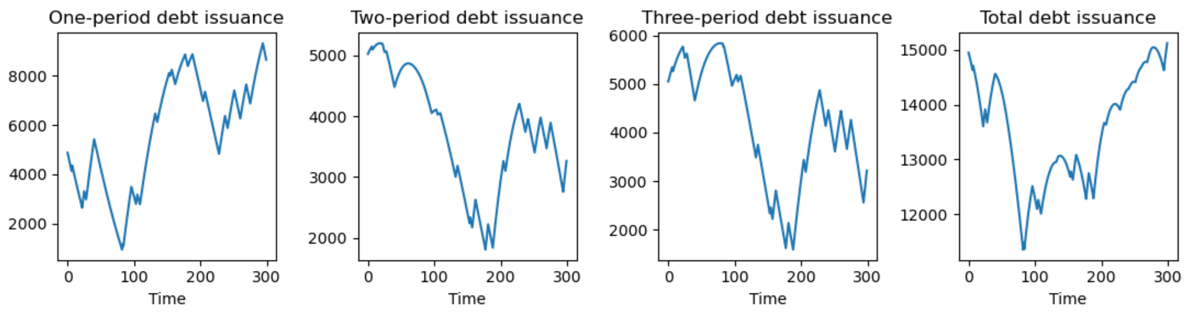
```
# New model parameters
H = 3
p1 = np.array([[0.9695], [0.902], [0.8369]])
p2 = np.array([[0.9295], [0.902], [0.8769]])
Pi = np.array([[0.9, 0.1], [0.1, 0.9]])

# Put penalty on different issuance across maturities
c2 = 0.5

A1, B1, C1, R1, Q1, W1 = LQ_markov_mapping_restruct(A22, C_2, Ug, H, p1, c2)
A2, B2, C2, R2, Q2, W2 = LQ_markov_mapping_restruct(A22, C_2, Ug, H, p2, c2)

# Small penalties on debt required to implement no-Ponzi scheme
R1[0, 0] = R1[0, 0] + 1e-9
R1[1, 1] = R1[1, 1] + 1e-9
R1[2, 2] = R1[2, 2] + 1e-9
R2[0, 0] = R2[0, 0] + 1e-9
R2[1, 1] = R2[1, 1] + 1e-9
R2[2, 2] = R2[2, 2] + 1e-9

# Construct lists of matrices
As = [A1, A2]
Bs = [B1, B2]
Cs = [C1, C2]
Rs = [R1, R2]
Qs = [Q1, Q2]
Ws = [W1, W2]

# Construct and solve the model using the LQMarkov class
lqm3 = qe.LQMarkov(Π, Qs, Rs, As, Bs, Cs=Cs, Ns=Ws, beta=β)
lqm3.stationary_values()

x0 = np.array([[5000, 5000, 5000, 1, 10]])
x, u, w, t = lqm3.compute_sequence(x0, ts_length=300)
```

```
# Plots of different maturities debt issuance

fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(11, 3))
ax1.plot(u[0, :])
ax1.set_title('One-period debt issuance')
ax1.set_xlabel('Time')
```

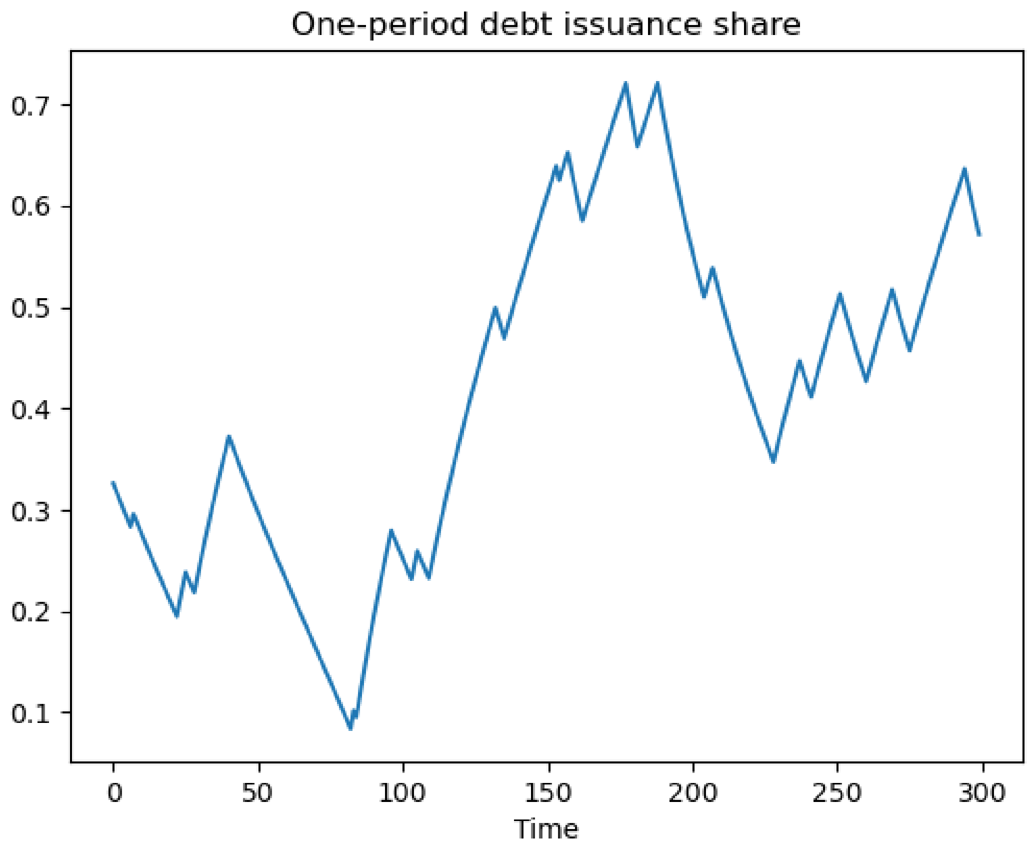
(continues on next page)

(continued from previous page)

```
ax2.plot(u[1, :])
ax2.set_title('Two-period debt issuance')
ax2.set_xlabel('Time')
ax3.plot(u[2, :])
ax3.set_title('Three-period debt issuance')
ax3.set_xlabel('Time')
ax4.plot(u[0, :] + u[1, :] + u[2, :])
ax4.set_title('Total debt issuance')
ax4.set_xlabel('Time')
plt.tight_layout()
plt.show()
```



```
# Plot share of debt issuance that is short-term

fig, ax = plt.subplots()
ax.plot((u[0, :] / (u[0, :] + u[1, :] + u[2, :])))
ax.set_title('One-period debt issuance share')
ax.set_xlabel('Time')
plt.show()
```


One-period debt issuance share
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250
300
Time

CHAPTER

SIXTEEN

HOW TO PAY FOR A WAR: PART 3

Contents

In addition to what's in Anaconda, this lecture deploys the quantecon library:

```
!pip install --upgrade quantecon
```

16.1 Another Application of Markov Jump Linear Quadratic Dynamic Programming

This is another *sequel to an earlier lecture*.

We again use a method introduced in lecture *Markov Jump LQ dynamic programming* to implement some ideas Barro (1999 [Barro, 1999], 2003 [Barro and McCleary, 2003]) that extend his classic 1979 [Barro, 1979] model of tax smoothing.

Barro's 1979 [Barro, 1979] model is about a government that borrows and lends in order to help it minimize an intertemporal measure of distortions caused by taxes.

Technically, Barro's 1979 [Barro, 1979] model looks a lot like a consumption-smoothing model.

Our generalizations of his 1979 model will also look like souped-up consumption-smoothing models.

In this lecture, we describe a tax-smoothing problem of a government that faces **roll-over risk**.

Let's start with some standard imports:

```
import quantecon as qe
import numpy as np
import matplotlib.pyplot as plt
```

16.2 Roll-Over Risk

Let T_t denote tax collections, β a discount factor, $b_{t,t+1}$ time $t+1$ goods that the government promises to pay at t, G_t government purchases, p^t_{t+1} the number of time t goods received per time $t+1$ goods promised.

The stochastic process of government expenditures is exogenous.

The government's problem is to choose a plan for borrowing and tax collections $\{b_{t+1}, T_t\}_{t=0}^\infty$ to minimize

$$E_0 \sum_{t=0}^\infty \beta^t T_t^2$$

subject to the constraints

$$T_t + p^t_{t+1} b_{t,t+1} = G_t + b_{t-1,t}$$

$$G_t = U_{g,t} z_t$$

$$z_{t+1} = A_{22,t} z_t + C_{2,t} w_{t+1}$$

where $w_{t+1} \sim N(0, I)$. The variables $T_t, b_{t,t+1}$ are *control* variables chosen at t, while $b_{t-1,t}$ is an endogenous state variable inherited from the past at time t and p^t_{t+1} is an exogenous state variable at time t.

This is the same set-up as used *in this lecture*.

We will consider a situation in which the government faces "roll-over risk".

Specifically, we shut down the government's ability to borrow in one of the Markov states.

16.3 A Dead End

A first thought for how to implement this might be to allow p^t_{t+1} to vary over time with:

$$p^t_{t+1} = \beta$$

in Markov state 1 and

$$p^t_{t+1} = 0$$

in Markov state 2.

Consequently, in the second Markov state, the government is unable to borrow, and the budget constraint becomes $T_t = G_t + b_{t-1,t}$.

However, if this is the only adjustment we make in our linear-quadratic model, the government will not set $b_{t,t+1} = 0$, which is the outcome we want to express *roll-over* risk in period t.

Instead, the government would have an incentive to set $b_{t,t+1}$ to a large negative number in state 2 – it would accumulate large amounts of *assets* to bring into period $t+1$ because that is cheap (Our Riccati equations will discover this for us!).

Thus, we must represent "roll-over risk" some other way.

16.4 Better Representation of Roll-Over Risk

To force the government to set $b_{t,t+1} = 0$, we can instead extend the model to have four Markov states:

1. Good today, good yesterday
2. Good today, bad yesterday
3. Bad today, good yesterday
4. Bad today, bad yesterday

where good is a state in which effectively the government can issue debt and bad is a state in which effectively the government can't issue debt.

We'll explain what *effectively* means shortly.

We now set

$$p^t_{t+1} = \beta$$

in all states.

In addition – and this is important because it defines what we mean by *effectively* – we put a large penalty on the $b_{t-1,t}$ element of the state vector in states 2 and 4.

This will prevent the government from wishing to issue any debt in states 3 or 4 because it would experience a large penalty from doing so in the next period.

The transition matrix for this formulation is:

$$\Pi = \begin{bmatrix} 0.95 & 0 & 0.05 & 0 \\ 0.95 & 0 & 0.05 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0 & 0.9 & 0 & 0.1 \end{bmatrix}$$

This transition matrix ensures that the Markov state cannot move, for example, from state 3 to state 1.

Because state 3 is "bad today", the next period cannot have "good yesterday".

```
# Model parameters
β, Gbar, ρ, σ = 0.95, 5, 0.8, 1

# Basic model matrices
A22 = np.array([[1, 0], [Gbar, ρ], ])
C2 = np.array([[0], [σ]])
Ug = np.array([[0, 1]])

# LQ framework matrices
A_t = np.zeros((1, 3))
A_b = np.hstack((np.zeros((2, 1)), A22))
A = np.vstack((A_t, A_b))

B = np.zeros((3, 1))
B[0, 0] = 1

C = np.vstack((np.zeros((1, 1)), C2))

Sg = np.hstack((np.zeros((1, 1)), Ug))
S1 = np.zeros((1, 3))
```

(continues on next page)

(continued from previous page)

```
S1[0, 0] = 1
S = S1 + Sg

R = S.T @ S

# Large penalty on debt in R2 to prevent borrowing in a bad state
R1 = np.copy(R)
R2 = np.copy(R)
R1[0, 0] = R[0, 0] + 1e-9
R2[0, 0] = R[0, 0] + 1e12

M = np.array([[-β]])
Q = M.T @ M
W = M.T @ S

Π = np.array([[0.95,   0, 0.05,   0],
              [0.95,   0, 0.05,   0],
              [0,    0.9,    0, 0.1],
              [0,    0.9,    0, 0.1]])

# Construct lists of matrices that correspond to each state
As = [A, A, A, A]
Bs = [B, B, B, B]
Cs = [C, C, C, C]
Rs = [R1, R2, R1, R2]
Qs = [Q, Q, Q, Q]
Ws = [W, W, W, W]

lqm = qe.LQMarkov(Π, Qs, Rs, As, Bs, Cs=Cs, Ns=Ws, beta=β)
lqm.stationary_values();
```

This model is simulated below, using the same process for G_t as in *this lecture*.

When $p^t_{t+1} = \beta$ government debt fluctuates around zero.

The spikes in the series for taxation show periods when the government is unable to access financial markets: positive spikes occur when debt is positive, and the government must raise taxes in the current period.

Negative spikes occur when the government has positive asset holdings.

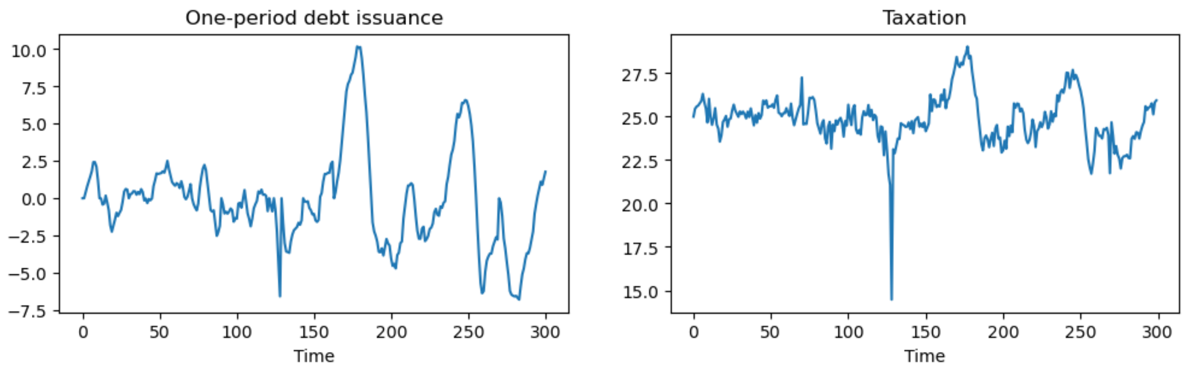
An inability to use financial markets in the next period means that the government uses those assets to lower taxation today.

```
x0 = np.array([[0, 1, 25]])
T = 300
x, u, w, state = lqm.compute_sequence(x0, ts_length=T)

# Calculate taxation each period from the budget constraint and the Markov state
tax = np.zeros([T, 1])
for i in range(T):
    tax[i, :] = S @ x[:, i] + M @ u[:, i]

# Plot of debt issuance and taxation
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 3))
ax1.plot(x[0, :])
ax1.set_title('One-period debt issuance')
ax1.set_xlabel('Time')
```

(continues on next page)

(continued from previous page)

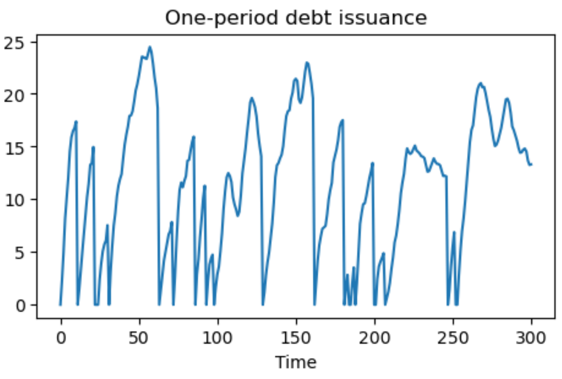
```
ax2.plot(tax)
ax2.set_title('Taxation')
ax2.set_xlabel('Time')
plt.show()
```

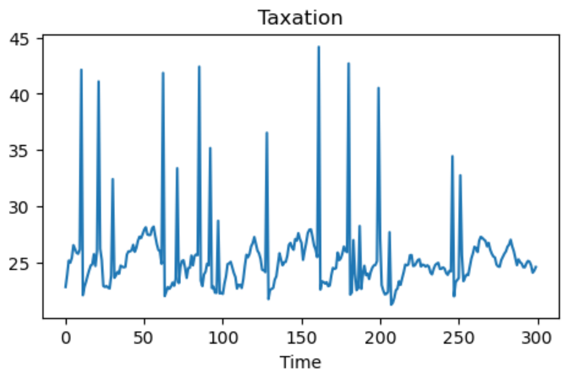


We can adjust the model so that, rather than having debt fluctuate around zero, the government is a debtor in every period we allow it to borrow.

To accomplish this, we simply raise p^t_{t+1} to $\beta + 0.02 = 0.97$.

```
M = np.array([[-β - 0.02]])

Q = M.T @ M
W = M.T @ S

# Construct lists of matrices
As = [A, A, A, A]
Bs = [B, B, B, B]
Cs = [C, C, C, C]
Rs = [R1, R2, R1, R2]
Qs = [Q, Q, Q, Q]
Ws = [W, W, W, W]

lqm2 = qe.LQMarkov(Π, Qs, Rs, As, Bs, Cs=Cs, Ns=Ws, beta=β)
x, u, w, state = lqm2.compute_sequence(x0, ts_length=T)

# Calculate taxation each period from the budget constraint and the
# Markov state
tax = np.zeros([T, 1])
for i in range(T):
    tax[i, :] = S @ x[:, i] + M @ u[:, i]

# Plot of debt issuance and taxation
fig, (ax1, ax2) =  plt.subplots(1, 2, figsize=(12, 3))
ax1.plot(x[0, :])
ax1.set_title('One-period debt issuance')
ax1.set_xlabel('Time')
ax2.plot(tax)
ax2.set_title('Taxation')
ax2.set_xlabel('Time')
plt.show()
```





With a lower interest rate, the government has an incentive to increase debt over time.

However, with “roll-over risk”, debt is recurrently reset to zero and taxes spike up.

Consequently, the government is wary of letting debt get too high, due to the high costs of a “sudden stop”.

CHAPTER

SEVENTEEN

OPTIMAL TAXATION IN AN LQ ECONOMY

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

17.1 Overview

In this lecture, we study optimal fiscal policy in a linear quadratic setting.

We modify a model of Robert Lucas and Nancy Stokey [Lucas and Stokey, 1983] so that convenient formulas for solving linear-quadratic models can be applied.

The economy consists of a representative household and a benevolent government.

The government finances an exogenous stream of government purchases with state-contingent loans and a linear tax on labor income.

A linear tax is sometimes called a flat-rate tax.

The household maximizes utility by choosing paths for consumption and labor, taking prices and the government's tax rate and borrowing plans as given.

Maximum attainable utility for the household depends on the government's tax and borrowing plans.

The *Ramsey problem* [Ramsey, 1927] is to choose tax and borrowing plans that maximize the household's welfare, taking the household's optimizing behavior as given.

There is a large number of competitive equilibria indexed by different government fiscal policies.

The Ramsey planner chooses the best competitive equilibrium.

We want to study the dynamics of tax rates, tax revenues, government debt under a Ramsey plan.

Because the Lucas and Stokey model features state-contingent government debt, the government debt dynamics differ substantially from those in a model of Robert Barro [Barro, 1979].

The treatment given here closely follows this manuscript, prepared by Thomas J. Sargent and Francois R. Velde.

We cover only the key features of the problem in this lecture, leaving you to refer to that source for additional results and intuition.

We'll need the following imports:

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from numpy import sqrt, eye, zeros, cumsum
from numpy.random import randn
import scipy.linalg
from collections import namedtuple
from quantecon import nullspace, mc_sample_path, var_quadratic_sum
```

17.1.1 Model Features

- Linear quadratic (LQ) model
- Representative household
- Stochastic dynamic programming over an infinite horizon
- Distortionary taxation

17.2 The Ramsey Problem

We begin by outlining the key assumptions regarding technology, households and the government sector.

17.2.1 Technology

Labor can be converted one-for-one into a single, non-storable consumption good.

In the usual spirit of the LQ model, the amount of labor supplied in each period is unrestricted.

This is unrealistic, but helpful when it comes to solving the model.

Realistic labor supply can be induced by suitable parameter values.

17.2.2 Households

Consider a representative household who chooses a path $\{\ell_t, c_t\}$ for labor and consumption to maximize

$$-\mathbb{E}\frac{1}{2}\sum_{t=0}^{\infty}\beta^t\left[(c_t-b_t)^2+\ell_t^2\right] \tag{17.1}$$

subject to the budget constraint

$$\mathbb{E}\sum_{t=0}^{\infty}\beta^t p_t^0\left[d_t+(1-\tau_t)\ell_t+s_t-c_t\right]=0 \tag{17.2}$$

Here

- β is a discount factor in $(0, 1)$.
- p_t^0 is a scaled Arrow-Debreu price at time 0 of history contingent goods at time $t + j$.
- b_t is a stochastic preference parameter.
- d_t is an endowment process.
- τ_t is a flat tax rate on labor income.
- s_t is a promised time-t coupon payment on debt issued by the government.

The scaled Arrow-Debreu price p_t^0 is related to the unscaled Arrow-Debreu price as follows.

If we let $\pi_t^0(x^t)$ denote the probability (density) of a history $x^t = [x_t, x_{t-1}, \ldots, x_0]$ of the state x^t, then the Arrow-Debreu time 0 price of a claim on one unit of consumption at date t, history x^t would be

$$\frac{\beta^t p_t^0}{\pi_t^0(x^t)}$$

Thus, our scaled Arrow-Debreu price is the ordinary Arrow-Debreu price multiplied by the discount factor β^t and divided by an appropriate probability.

The budget constraint (17.2) requires that the present value of consumption be restricted to equal the present value of endowments, labor income and coupon payments on bond holdings.

17.2.3 Government

The government imposes a linear tax on labor income, fully committing to a stochastic path of tax rates at time zero.

The government also issues state-contingent debt.

Given government tax and borrowing plans, we can construct a competitive equilibrium with distorting government taxes.

Among all such competitive equilibria, the Ramsey plan is the one that maximizes the welfare of the representative consumer.

17.2.4 Exogenous Variables

Endowments, government expenditure, the preference shock process b_t, and promised coupon payments on initial government debt s_t are all exogenous, and given by

- $d_t = S_d x_t$
- $g_t = S_g x_t$
- $b_t = S_b x_t$
- $s_t = S_s x_t$

The matrices S_d, S_g, S_b, S_s are primitives and $\{x_t\}$ is an exogenous stochastic process taking values in $\mathbb{R}^k$.

We consider two specifications for $\{x_t\}$.

1. Discrete case: $\{x_t\}$ is a discrete state Markov chain with transition matrix P.
2. VAR case: $\{x_t\}$ obeys $x_{t+1} = Ax_t + Cw_{t+1}$ where $\{w_t\}$ is independent zero-mean Gaussian with identify covariance matrix.

17.2.5 Feasibility

The period-by-period feasibility restriction for this economy is

$$c_t + g_t = d_t + \ell_t \tag{17.3}$$

A labor-consumption process $\{\ell_t, c_t\}$ is called *feasible* if (17.3) holds for all t.

17.2.6 Government Budget Constraint

Where p_t^0 is again a scaled Arrow-Debreu price, the time zero government budget constraint is

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t p_t^0 (s_t + g_t - \tau_t \ell_t) = 0 \tag{17.4}$$

17.2.7 Equilibrium

An *equilibrium* is a feasible allocation $\{\ell_t, c_t\}$, a sequence of prices $\{p_t^0\}$, and a tax system $\{\tau_t\}$ such that

1. The allocation $\{\ell_t, c_t\}$ is optimal for the household given $\{p_t^0\}$ and $\{\tau_t\}$.
2. The government's budget constraint (17.4) is satisfied.

The *Ramsey problem* is to choose the equilibrium $\{\ell_t, c_t, \tau_t, p_t^0\}$ that maximizes the household's welfare.

If $\{\ell_t, c_t, \tau_t, p_t^0\}$ solves the Ramsey problem, then $\{\tau_t\}$ is called the *Ramsey plan*.

The solution procedure we adopt is

1. Use the first-order conditions from the household problem to pin down prices and allocations given $\{\tau_t\}$.
2. Use these expressions to rewrite the government budget constraint (17.4) in terms of exogenous variables and allocations.
3. Maximize the household's objective function (17.1) subject to the constraint constructed in step 2 and the feasibility constraint (17.3).

The solution to this maximization problem pins down all quantities of interest.

17.2.8 Solution

Step one is to obtain the first-conditions for the household's problem, taking taxes and prices as given.

Letting μ be the Lagrange multiplier on (17.2), the first-order conditions are $p_t^0 = (c_t - b_t)/\mu$ and $\ell_t = (c_t - b_t)(1 - \tau_t)$.

Rearranging and normalizing at $\mu = b_0 - c_0$, we can write these conditions as

$$p_t^0 = \frac{b_t - c_t}{b_0 - c_0} \quad \text{and} \quad \tau_t = 1 - \frac{\ell_t}{b_t - c_t} \tag{17.5}$$

Substituting (17.5) into the government's budget constraint (17.4) yields

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t \left[(b_t - c_t)(s_t + g_t - \ell_t) + \ell_t^2 \right] = 0 \tag{17.6}$$

The Ramsey problem now amounts to maximizing (17.1) subject to (17.6) and (17.3).

The associated Lagrangian is

$$\mathcal{L} = \mathbb{E} \sum_{t=0}^{\infty} \beta^t \left\{ -\frac{1}{2} \left[(c_t - b_t)^2 + \ell_t^2 \right] + \lambda \left[(b_t - c_t)(\ell_t - s_t - g_t) - \ell_t^2 \right] + \mu_t [d_t + \ell_t - c_t - g_t] \right\} \tag{17.7}$$

The first-order conditions associated with c_t and ℓ_t are

$$-(c_t - b_t) + \lambda[-\ell_t + (g_t + s_t)] = \mu_t$$

and

$$\ell_t - \lambda[(b_t - c_t) - 2\ell_t] = \mu_t$$

Combining these last two equalities with (17.3) and working through the algebra, one can show that

$$\ell_t = \bar{\ell}_t - \nu m_t \quad \text{and} \quad c_t = \bar{c}_t - \nu m_t \tag{17.8}$$

where

- $\nu := \lambda/(1 + 2\lambda)$
- $\bar{\ell}_t := (b_t - d_t + g_t)/2$
- $\bar{c}_t := (b_t + d_t - g_t)/2$
- $m_t := (b_t - d_t - s_t)/2$

Apart from ν, all of these quantities are expressed in terms of exogenous variables.

To solve for ν, we can use the government's budget constraint again.

The term inside the brackets in (17.6) is $(b_t - c_t)(s_t + g_t) - (b_t - c_t)\ell_t + \ell_t^2$.

Using (17.8), the definitions above and the fact that $\bar{\ell} = b - \bar{c}$, this term can be rewritten as

$$(b_t - \bar{c}_t)(g_t + s_t) + 2m_t^2(\nu^2 - \nu)$$

Reinserting into (17.6), we get

$$\mathbb{E} \left\{ \sum_{t=0}^{\infty} \beta^t (b_t - \bar{c}_t)(g_t + s_t) \right\} + (\nu^2 - \nu) \mathbb{E} \left\{ \sum_{t=0}^{\infty} \beta^t 2 m_t^2 \right\} = 0 \tag{17.9}$$

Although it might not be clear yet, we are nearly there because:

- The two expectations terms in (17.9) can be solved for in terms of model primitives.
- This in turn allows us to solve for the Lagrange multiplier ν.
- With ν in hand, we can go back and solve for the allocations via (17.8).
- Once we have the allocations, prices and the tax system can be derived from (17.5).

17.2.9 Computing the Quadratic Term

Let's consider how to obtain the term ν in (17.9).

If we can compute the two expected geometric sums

$$b_0 := \mathbb{E} \left\{ \sum_{t=0}^{\infty} \beta^t (b_t - \bar{c}_t)(g_t + s_t) \right\} \quad \text{and} \quad a_0 := \mathbb{E} \left\{ \sum_{t=0}^{\infty} \beta^t 2 m_t^2 \right\} \tag{17.10}$$

then the problem reduces to solving

$$b_0 + a_0(\nu^2 - \nu) = 0$$

for ν.

Provided that $4b_0 < a_0$, there is a unique solution $\nu \in (0, 1/2)$, and a unique corresponding $\lambda > 0$.

Let's work out how to compute mathematical expectations in (17.10).

For the first one, the random variable $(b_t - \bar{c}_t)(g_t + s_t)$ inside the summation can be expressed as

$$\frac{1}{2} x_t'(S_b - S_d + S_g)'(S_g + S_s)x_t$$

For the second expectation in (17.10), the random variable $2m_t^2$ can be written as

$$\frac{1}{2} x_t'(S_b - S_d - S_s)'(S_b - S_d - S_s)x_t$$

It follows that both objects of interest are special cases of the expression

$$q(x_0) = \mathbb{E} \sum_{t=0}^{\infty} \beta^t x_t' H x_t \tag{17.11}$$

where H is a matrix conformable to x_t and x_t' is the transpose of column vector x_t.

Suppose first that $\{x_t\}$ is the Gaussian VAR described *above*.

In this case, the formula for computing $q(x_0)$ is known to be $q(x_0) = x_0' Q x_0 + v$, where

- Q is the solution to $Q = H + \beta A' Q A$, and
- $v = \text{trace}\,(C' Q C) \beta / (1 - \beta)$

The first equation is known as a discrete Lyapunov equation and can be solved using this function.

17.2.10 Finite State Markov Case

Next, suppose that $\{x_t\}$ is the discrete Markov process described *above*.

Suppose further that each x_t takes values in the state space $\{x^1, \dots, x^N\} \subset \mathbb{R}^k$.

Let $h \colon \mathbb{R}^k \to \mathbb{R}$ be a given function, and suppose that we wish to evaluate

$$q(x_0) = \mathbb{E} \sum_{t=0}^{\infty} \beta^t h(x_t) \quad \text{given} \quad x_0 = x^j$$

For example, in the discussion above, $h(x_t) = x_t' H x_t$.

It is legitimate to pass the expectation through the sum, leading to

$$q(x_0) = \sum_{t=0}^{\infty} \beta^t (P^t h)[j] \tag{17.12}$$

Here

- P^t is the t-th power of the transition matrix P.
- h is, with some abuse of notation, the vector $(h(x^1), \dots, h(x^N))$.
- $(P^t h)[j]$ indicates the j-th element of $P^t h$.

It can be shown that (17.12) is in fact equal to the j-th element of the vector $(I - \beta P)^{-1} h$.

This last fact is applied in the calculations below.

17.2.11 Other Variables

We are interested in tracking several other variables besides the ones described above.

To prepare the way for this, we define

$$p_{t+j}^t = \frac{b_{t+j} - c_{t+j}}{b_t - c_t}$$

as the scaled Arrow-Debreu time t price of a history contingent claim on one unit of consumption at time $t + j$.

These are prices that would prevail at time t if markets were reopened at time t.

These prices are constituents of the present value of government obligations outstanding at time t, which can be expressed as

$$B_t := \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j p_{t+j}^t (\tau_{t+j} \ell_{t+j} - g_{t+j}) \tag{17.13}$$

Using our expression for prices and the Ramsey plan, we can also write B_t as

$$B_t = \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j \frac{(b_{t+j} - c_{t+j})(\ell_{t+j} - g_{t+j}) - \ell_{t+j}^2}{b_t - c_t}$$

This version is more convenient for computation.

Using the equation

$$p_{t+j}^t = p_{t+1}^t p_{t+j}^{t+1}$$

it is possible to verify that (17.13) implies that

$$B_t = (\tau_t \ell_t - g_t) + E_t \sum_{j=1}^{\infty} p_{t+j}^t (\tau_{t+j} \ell_{t+j} - g_{t+j})$$

and

$$B_t = (\tau_t \ell_t - g_t) + \beta E_t p_{t+1}^t B_{t+1} \tag{17.14}$$

Define

$$R_t^{-1} := \mathbb{E}_t \beta^j p_{t+1}^t \tag{17.15}$$

R_t is the gross 1-period risk-free rate for loans between t and $t + 1$.

17.2.12 A Martingale

We now want to study the following two objects, namely,

$$\pi_{t+1} := B_{t+1} - R_t [B_t - (\tau_t \ell_t - g_t)]$$

and the cumulation of π_t

$$\Pi_t := \sum_{s=0}^{t} \pi_t$$

The term π_{t+1} is the difference between two quantities:

- B_{t+1}, the value of government debt at the start of period $t+1$.
- $R_t[B_t + g_t - \tau_t]$, which is what the government would have owed at the beginning of period $t+1$ if it had simply borrowed at the one-period risk-free rate rather than selling state-contingent securities.

Thus, π_{t+1} is the excess payout on the actual portfolio of state-contingent government debt relative to an alternative portfolio sufficient to finance $B_t + g_t - \tau_t \ell_t$ and consisting entirely of risk-free one-period bonds.

Use expressions (17.14) and (17.15) to obtain

$$\pi_{t+1} = B_{t+1} - \frac{1}{\beta E_t p^t_{t+1}} \left[\beta E_t p^t_{t+1} B_{t+1}\right]$$

or

$$\pi_{t+1} = B_{t+1} - \tilde{E}_t B_{t+1} \tag{17.16}$$

where $\tilde{E}_t$ is the conditional mathematical expectation taken with respect to a one-step transition density that has been formed by multiplying the original transition density with the likelihood ratio

$$m^t_{t+1} = \frac{p^t_{t+1}}{E_t p^t_{t+1}}$$

It follows from equation (17.16) that

$$\tilde{E}_t \pi_{t+1} = \tilde{E}_t B_{t+1} - \tilde{E}_t B_{t+1} = 0$$

which asserts that $\{\pi_{t+1}\}$ is a martingale difference sequence under the distorted probability measure, and that $\{\Pi_t\}$ is a martingale under the distorted probability measure.

In the tax-smoothing model of Robert Barro [Barro, 1979], government debt is a random walk.

In the current model, government debt $\{B_t\}$ is not a random walk, but the `excess payoff` $\{\Pi_t\}$ on it is.

17.3 Implementation

The following code provides functions for

1. Solving for the Ramsey plan given a specification of the economy.
2. Simulating the dynamics of the major variables.

Description and clarifications are given below

```python
# Set up a namedtuple to store data on the model economy
Economy = namedtuple('economy',
                     ('β',          # Discount factor
                     'Sg',          # Govt spending selector matrix
                     'Sd',          # Exogenous endowment selector matrix
                     'Sb',          # Utility parameter selector matrix
                     'Ss',          # Coupon payments selector matrix
                     'discrete',    # Discrete or continuous -- boolean
                     'proc'))       # Stochastic process parameters

# Set up a namedtuple to store return values for compute_paths()
Path = namedtuple('path',
                 ('g',             # Govt spending
                 'd',              # Endowment
```

(continues on next page)

(continued from previous page)

```
                 'b',                # Utility shift parameter
                 's',                # Coupon payment on existing debt
                 'c',                # Consumption
                 'l',                # Labor
                 'p',                # Price
                 'τ',                # Tax rate
                 'rvn',              # Revenue
                 'B',                # Govt debt
                 'R',                # Risk-free gross return
                 'π',                # One-period risk-free interest rate
                 'Π',                # Cumulative rate of return, adjusted
                 'ξ'))               # Adjustment factor for Π


def compute_paths(T, econ):
    """
    Compute simulated time paths for exogenous and endogenous variables.

    Parameters
    ===========
    T: int
        Length of the simulation

    econ: a namedtuple of type 'Economy', containing
        β           - Discount factor
        Sg          - Govt spending selector matrix
        Sd          - Exogenous endowment selector matrix
        Sb          - Utility parameter selector matrix
        Ss          - Coupon payments selector matrix
        discrete    - Discrete exogenous process (True or False)
        proc        - Stochastic process parameters

    Returns
    ========
    path: a namedtuple of type 'Path', containing
        g             - Govt spending
        d             - Endowment
        b             - Utility shift parameter
        s             - Coupon payment on existing debt
        c             - Consumption
        l             - Labor
        p             - Price
        τ             - Tax rate
        rvn           - Revenue
        B             - Govt debt
        R             - Risk-free gross return
        π             - One-period risk-free interest rate
        Π             - Cumulative rate of return, adjusted
        ξ             - Adjustment factor for Π

        The corresponding values are flat numpy ndarrays.

    """

    # Simplify names
    β, Sg, Sd, Sb, Ss = econ.β, econ.Sg, econ.Sd, econ.Sb, econ.Ss
```

(continues on next page)

(continued from previous page)

```
    if econ.discrete:
        P, x_vals = econ.proc
    else:
        A, C = econ.proc

    # Simulate the exogenous process x
    if econ.discrete:
        state = mc_sample_path(P, init=0, sample_size=T)
        x = x_vals[:, state]
    else:
        # Generate an initial condition x0 satisfying x0 = A x0
        nx, nx = A.shape
        x0 = nullspace((eye(nx) - A))
        x0 = -x0 if (x0[nx-1] < 0) else x0
        x0 = x0 / x0[nx-1]

        # Generate a time series x of length T starting from x0
        nx, nw = C.shape
        x = zeros((nx, T))
        w = randn(nw, T)
        x[:, 0] = x0.T
        for t in range(1, T):
            x[:, t] = A @ x[:, t-1] + C @ w[:, t]

    # Compute exogenous variable sequences
    g, d, b, s = ((S @ x).flatten() for S in (Sg, Sd, Sb, Ss))

    # Solve for Lagrange multiplier in the govt budget constraint
    # In fact we solve for v = lambda / (1 + 2*lambda).  Here v is the
    # solution to a quadratic equation a(v**2 - v) + b = 0 where
    # a and b are expected discounted sums of quadratic forms of the state.
    Sm = Sb - Sd - Ss
    # Compute a and b
    if econ.discrete:
        ns = P.shape[0]
        F = scipy.linalg.inv(eye(ns) - β * P)
        a0 = 0.5 * (F @ (x_vals.T @ Sm.T)**2)[0]
        H = ((Sb - Sd + Sg) @ x_vals) * ((Sg - Ss) @ x_vals)
        b0 = 0.5 * (F @ H.T)[0]
        a0, b0 = float(a0), float(b0)
    else:
        H = Sm.T @ Sm
        a0 = 0.5 * var_quadratic_sum(A, C, H, β, x0)
        H = (Sb - Sd + Sg).T @ (Sg + Ss)
        b0 = 0.5 * var_quadratic_sum(A, C, H, β, x0)

    # Test that v has a real solution before assigning
    warning_msg = """
    Hint: you probably set government spending too {}.  Elect a {}
    Congress and start over.
    """
    disc = a0**2 - 4 * a0 * b0
    if disc >= 0:
        v = 0.5 * (a0 - sqrt(disc)) / a0
    else:
```

(continues on next page)

(continued from previous page)

```
        print("There is no Ramsey equilibrium for these parameters.")
        print(warning_msg.format('high', 'Republican'))
        sys.exit(0)

    # Test that the Lagrange multiplier has the right sign
    if ν * (0.5 - ν) < 0:
        print("Negative multiplier on the government budget constraint.")
        print(warning_msg.format('low', 'Democratic'))
        sys.exit(0)

    # Solve for the allocation given ν and x
    Sc = 0.5 * (Sb + Sd - Sg - ν * Sm)
    Sl = 0.5 * (Sb - Sd + Sg - ν * Sm)
    c = (Sc @ x).flatten()
    l = (Sl @ x).flatten()
    p = ((Sb - Sc) @ x).flatten()  # Price without normalization
    τ = 1 - l / (b - c)
    rvn = l * τ

    # Compute remaining variables
    if econ.discrete:
        H = ((Sb - Sc) @ x_vals) * ((Sl - Sg) @ x_vals) - (Sl @ x_vals)**2
        temp = (F @ H.T).flatten()
        B = temp[state] / p
        H = (P[state, :] @ x_vals.T @ (Sb - Sc).T).flatten()
        R = p / (β * H)
        temp = ((P[state, :] @ x_vals.T @ (Sb - Sc).T)).flatten()
        ξ = p[1:] / temp[:T-1]
    else:
        H = Sl.T @ Sl - (Sb - Sc).T @ (Sl - Sg)
        L = np.empty(T)
        for t in range(T):
            L[t] = var_quadratic_sum(A, C, H, β, x[:, t])
        B = L / p
        Rinv = (β * ((Sb - Sc) @ A @ x)).flatten() / p
        R = 1 / Rinv
        AF1 = (Sb - Sc) @ x[:, 1:]
        AF2 = (Sb - Sc) @ A @ x[:, :T-1]
        ξ = AF1 / AF2
        ξ = ξ.flatten()

    π = B[1:] - R[:T-1] * B[:T-1] - rvn[:T-1] + g[:T-1]
    Π = cumsum(π * ξ)

    # Prepare return values
    path = Path(g=g, d=d, b=b, s=s, c=c, l=l, p=p,
                τ=τ, rvn=rvn, B=B, R=R, π=π, Π=Π, ξ=ξ)

    return path


def gen_fig_1(path):
    """
    The parameter is the path namedtuple returned by compute_paths().  See
    the docstring of that function for details.
    """
```

(continues on next page)

(continued from previous page)

```
    T = len(path.c)

    # Prepare axes
    num_rows, num_cols = 2, 2
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(14, 10))
    plt.subplots_adjust(hspace=0.4)
    for i in range(num_rows):
        for j in range(num_cols):
            axes[i, j].grid()
            axes[i, j].set_xlabel('Time')
    bbox = (0., 1.02, 1., .102)
    legend_args = {'bbox_to_anchor': bbox, 'loc': 3, 'mode': 'expand'}
    p_args = {'lw': 2, 'alpha': 0.7}

    # Plot consumption, govt expenditure and revenue
    ax = axes[0, 0]
    ax.plot(path.rvn, label=r'$\tau_t \ell_t$', **p_args)
    ax.plot(path.g, label='$g_t$', **p_args)
    ax.plot(path.c, label='$c_t$', **p_args)
    ax.legend(ncol=3, **legend_args)

    # Plot govt expenditure and debt
    ax = axes[0, 1]
    ax.plot(list(range(1, T+1)), path.rvn, label=r'$\tau_t \ell_t$', **p_args)
    ax.plot(list(range(1, T+1)), path.g, label='$g_t$', **p_args)
    ax.plot(list(range(1, T)), path.B[1:T], label='$B_{t+1}$', **p_args)
    ax.legend(ncol=3, **legend_args)

    # Plot risk-free return
    ax = axes[1, 0]
    ax.plot(list(range(1, T+1)), path.R - 1, label='$R_t - 1$', **p_args)
    ax.legend(ncol=1, **legend_args)

    # Plot revenue, expenditure and risk free rate
    ax = axes[1, 1]
    ax.plot(list(range(1, T+1)), path.rvn, label=r'$\tau_t \ell_t$', **p_args)
    ax.plot(list(range(1, T+1)), path.g, label='$g_t$', **p_args)
    axes[1, 1].plot(list(range(1, T)), path.π, label=r'$\pi_{t+1}$', **p_args)
    ax.legend(ncol=3, **legend_args)

    plt.show()


def gen_fig_2(path):
    """
    The parameter is the path namedtuple returned by compute_paths(). See
    the docstring of that function for details.
    """

    T = len(path.c)

    # Prepare axes
    num_rows, num_cols = 2, 1
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 10))
    plt.subplots_adjust(hspace=0.5)
```

(continues on next page)

(continued from previous page)

```
    bbox = (0., 1.02, 1., .102)
    bbox = (0., 1.02, 1., .102)
    legend_args = {'bbox_to_anchor': bbox, 'loc': 3, 'mode': 'expand'}
    p_args = {'lw': 2, 'alpha': 0.7}

    # Plot adjustment factor
    ax = axes[0]
    ax.plot(list(range(2, T+1)), path.ξ, label=r'$\xi_t$', **p_args)
    ax.grid()
    ax.set_xlabel('Time')
    ax.legend(ncol=1, **legend_args)

    # Plot adjusted cumulative return
    ax = axes[1]
    ax.plot(list(range(2, T+1)), path.Π, label=r'$\Pi_t$', **p_args)
    ax.grid()
    ax.set_xlabel('Time')
    ax.legend(ncol=1, **legend_args)

    plt.show()
```

17.3.1 Comments on the Code

The function `var_quadratic_sum` imported from `quadsums` is for computing the value of (17.11) when the exogenous process $\{x_t\}$ is of the VAR type described *above*.

Below the definition of the function, you will see definitions of two `namedtuple` objects, `Economy` and `Path`.

The first is used to collect all the parameters and primitives of a given LQ economy, while the second collects output of the computations.

In Python, a `namedtuple` is a popular data type from the `collections` module of the standard library that replicates the functionality of a tuple, but also allows you to assign a name to each tuple element.

These elements can then be references via dotted attribute notation — see for example the use of `path` in the functions `gen_fig_1()` and `gen_fig_2()`.

The benefits of using `namedtuples`:

- Keeps content organized by meaning.
- Helps reduce the number of global variables.

Other than that, our code is long but relatively straightforward.

17.4 Examples

Let's look at two examples of usage.

17.4.1 The Continuous Case

Our first example adopts the VAR specification described *above*.

Regarding the primitives, we set

- $\beta = 1/1.05$
- $b_t = 2.135$ and $s_t = d_t = 0$ for all t

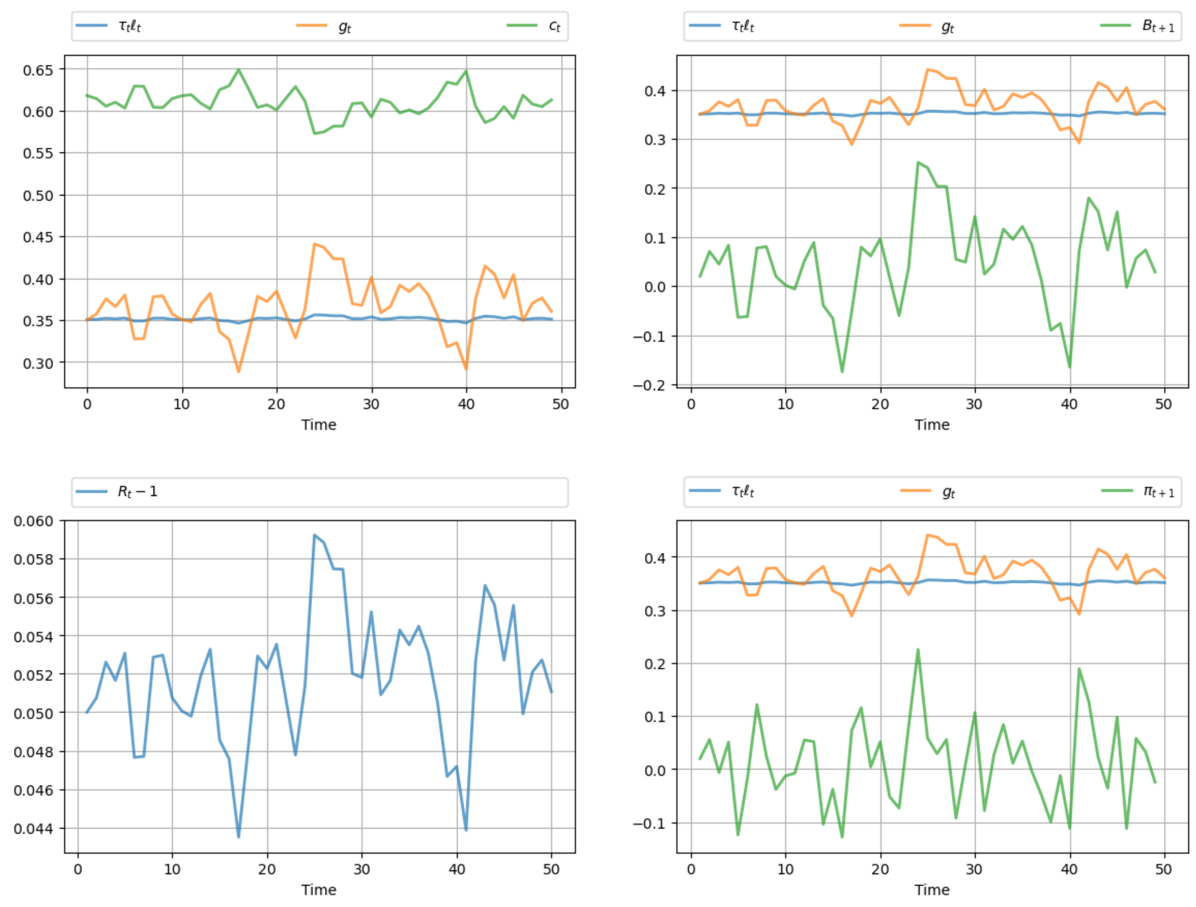
Government spending evolves according to

$$g_{t+1} - \mu_g = \rho(g_t - \mu_g) + C_g w_{g,t+1}$$

with $\rho = 0.7$, $\mu_g = 0.35$ and $C_g = \mu_g \sqrt{1-\rho^2}/10$.

Here's the code

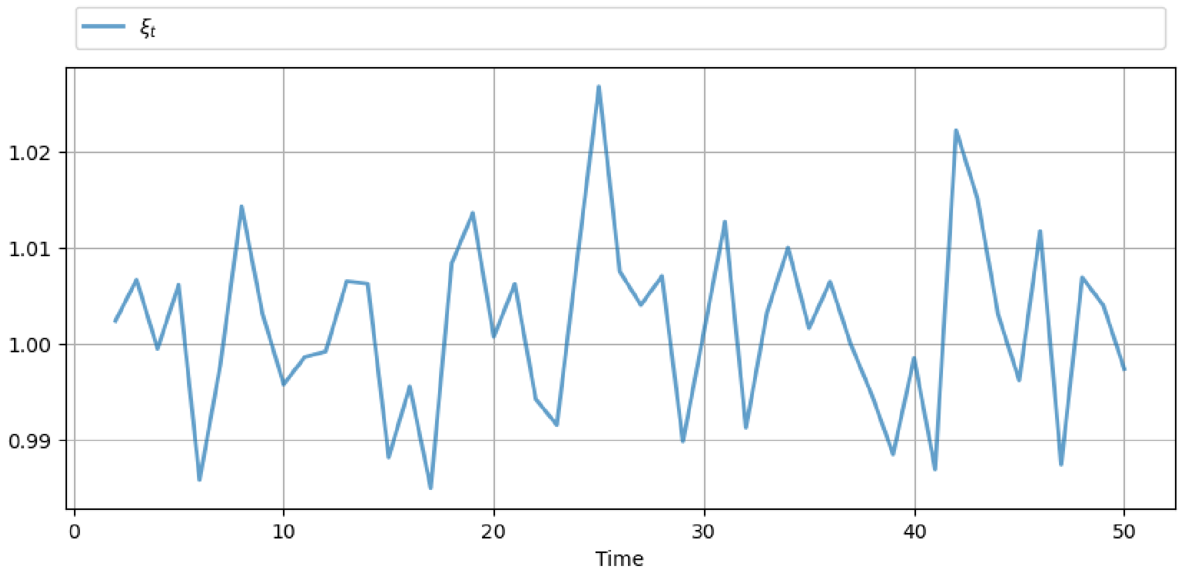
```
# == Parameters == #
β = 1 / 1.05
ρ, mg = .7, .35
A = eye(2)
A[0, :] = ρ, mg * (1-ρ)
C = np.zeros((2, 1))
C[0, 0] = np.sqrt(1 - ρ**2) * mg / 10
Sg = np.array((1, 0)).reshape(1, 2)
Sd = np.array((0, 0)).reshape(1, 2)
Sb = np.array((0, 2.135)).reshape(1, 2)
Ss = np.array((0, 0)).reshape(1, 2)

economy = Economy(β=β, Sg=Sg, Sd=Sd, Sb=Sb, Ss=Ss,
                  discrete=False, proc=(A, C))

T = 50
path = compute_paths(T, economy)
gen_fig_1(path)
```

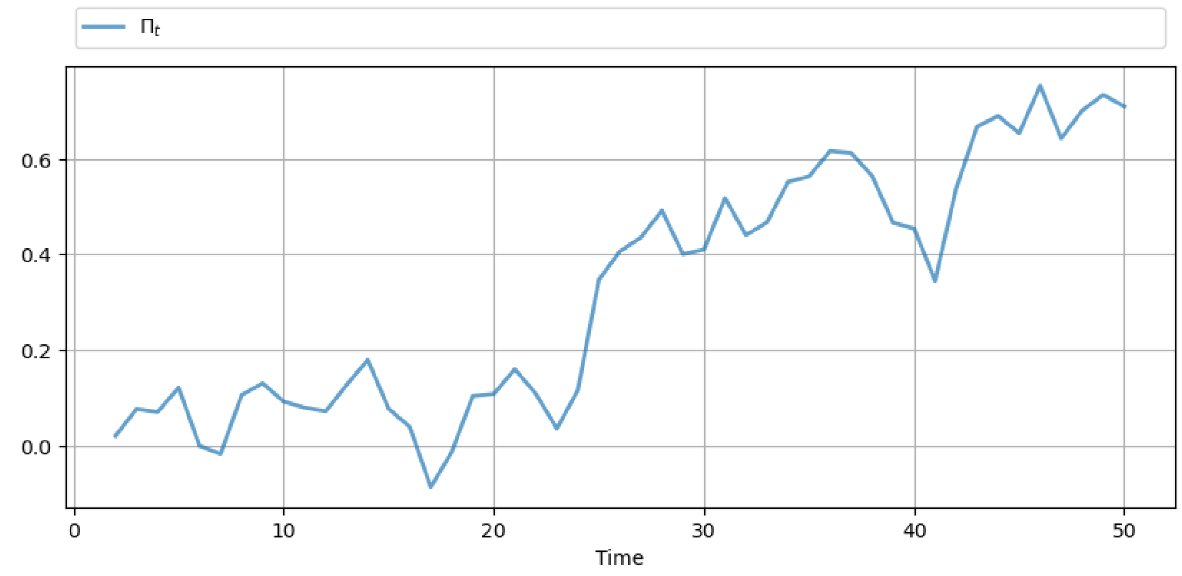


The legends on the figures indicate the variables being tracked.

Most obvious from the figure is tax smoothing in the sense that tax revenue is much less variable than government expenditure.

```
gen_fig_2(path)
```





See the original manuscript for comments and interpretation.

17.4.2 The Discrete Case

Our second example adopts a discrete Markov specification for the exogenous process

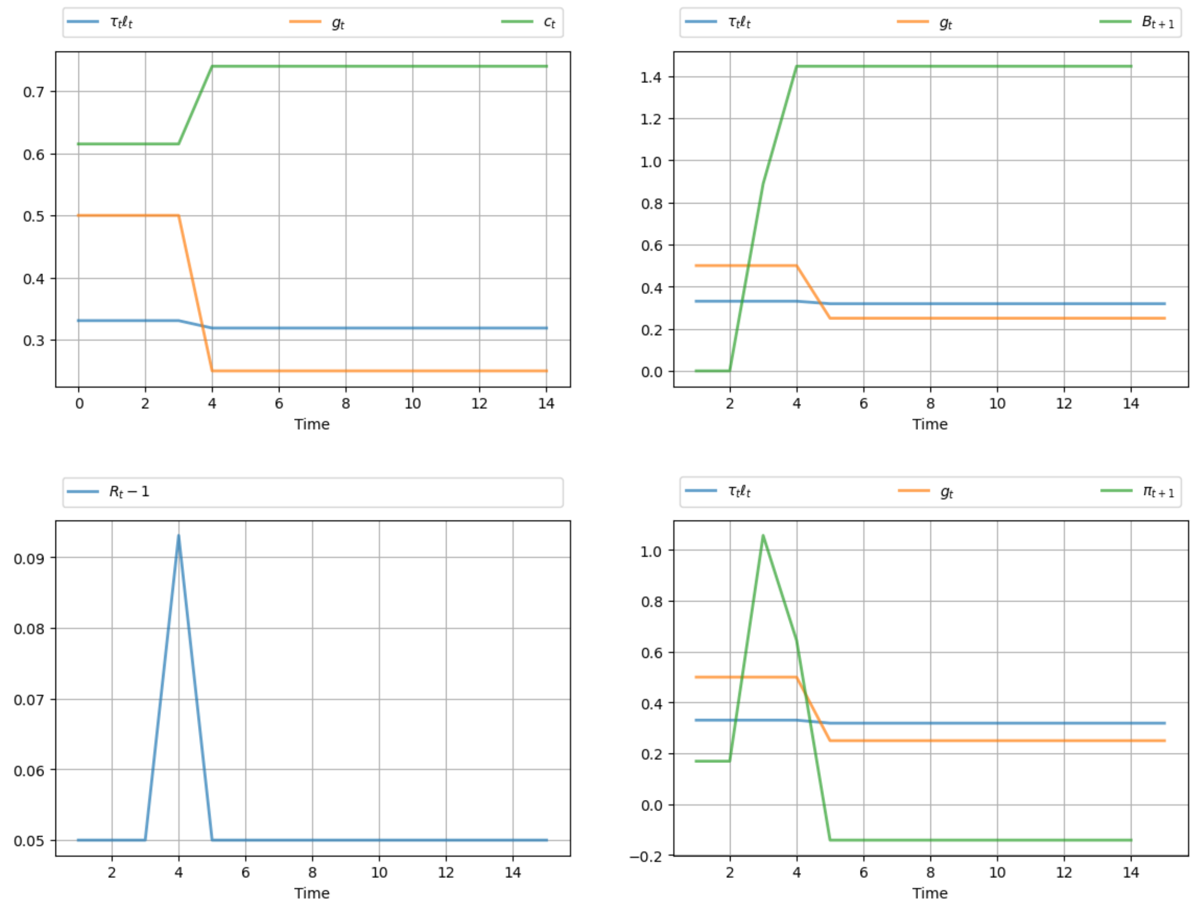
```
# == Parameters == #
β = 1 / 1.05
P = np.array([[0.8, 0.2, 0.0],
              [0.0, 0.5, 0.5],
              [0.0, 0.0, 1.0]])

# Possible states of the world
```

(continues on next page)

(continued from previous page)

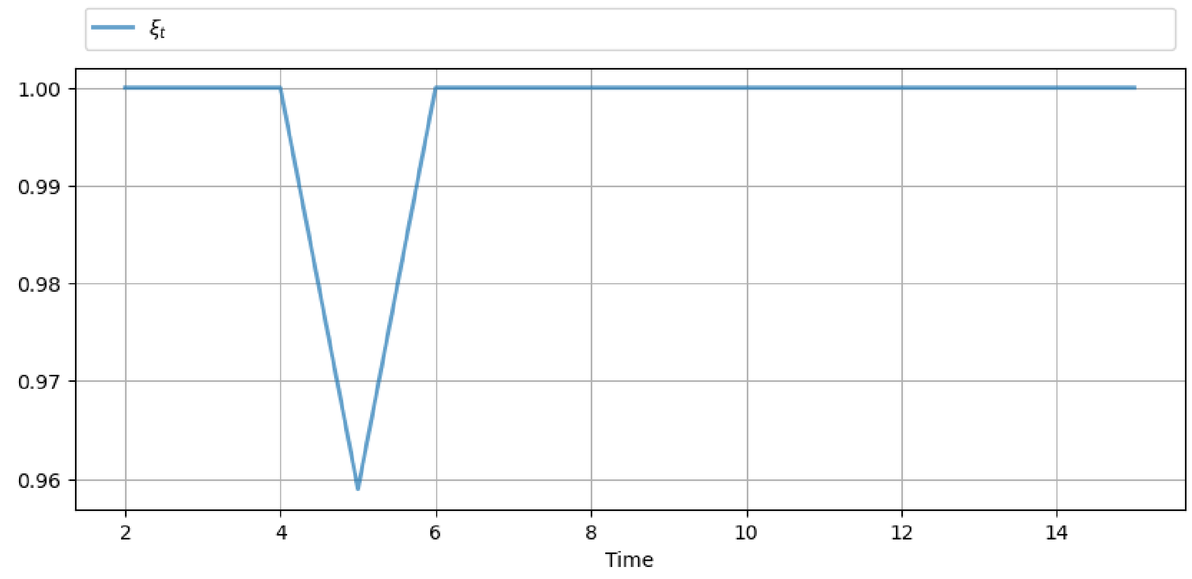
```
# Each column is a state of the world. The rows are [g d b s 1]
x_vals = np.array([[0.5, 0.5, 0.25],
                  [0.0, 0.0,  0.0],
                  [2.2, 2.2,  2.2],
                  [0.0, 0.0,  0.0],
                  [1.0, 1.0,  1.0]])

Sg = np.array((1, 0, 0, 0, 0)).reshape(1, 5)
Sd = np.array((0, 1, 0, 0, 0)).reshape(1, 5)
Sb = np.array((0, 0, 1, 0, 0)).reshape(1, 5)
Ss = np.array((0, 0, 0, 1, 0)).reshape(1, 5)

economy = Economy(β=β, Sg=Sg, Sd=Sd, Sb=Sb, Ss=Ss,
                  discrete=True, proc=(P, x_vals))

T = 15
path = compute_paths(T, economy)
gen_fig_1(path)
```

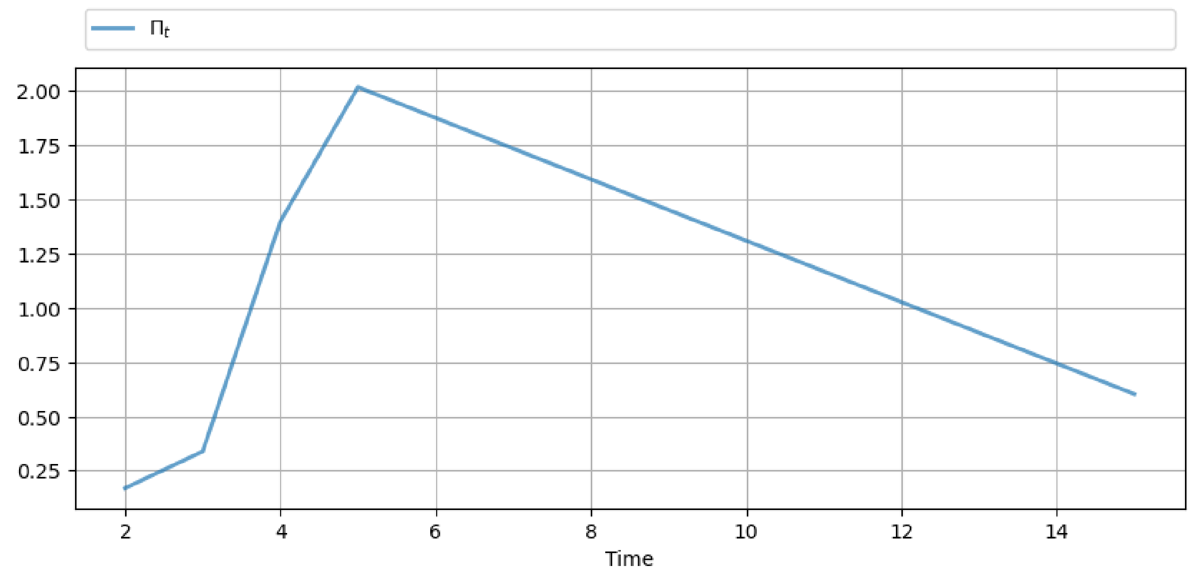
```
/tmp/ipykernel_6535/2748685684.py:111: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  a0, b0 = float(a0), float(b0)
```



The call `gen_fig_2(path)` generates

```
gen_fig_2(path)
```





See the original manuscript for comments and interpretation.

17.5 Exercises

Exercise 17.5.1

Modify the VAR example *given above*, setting

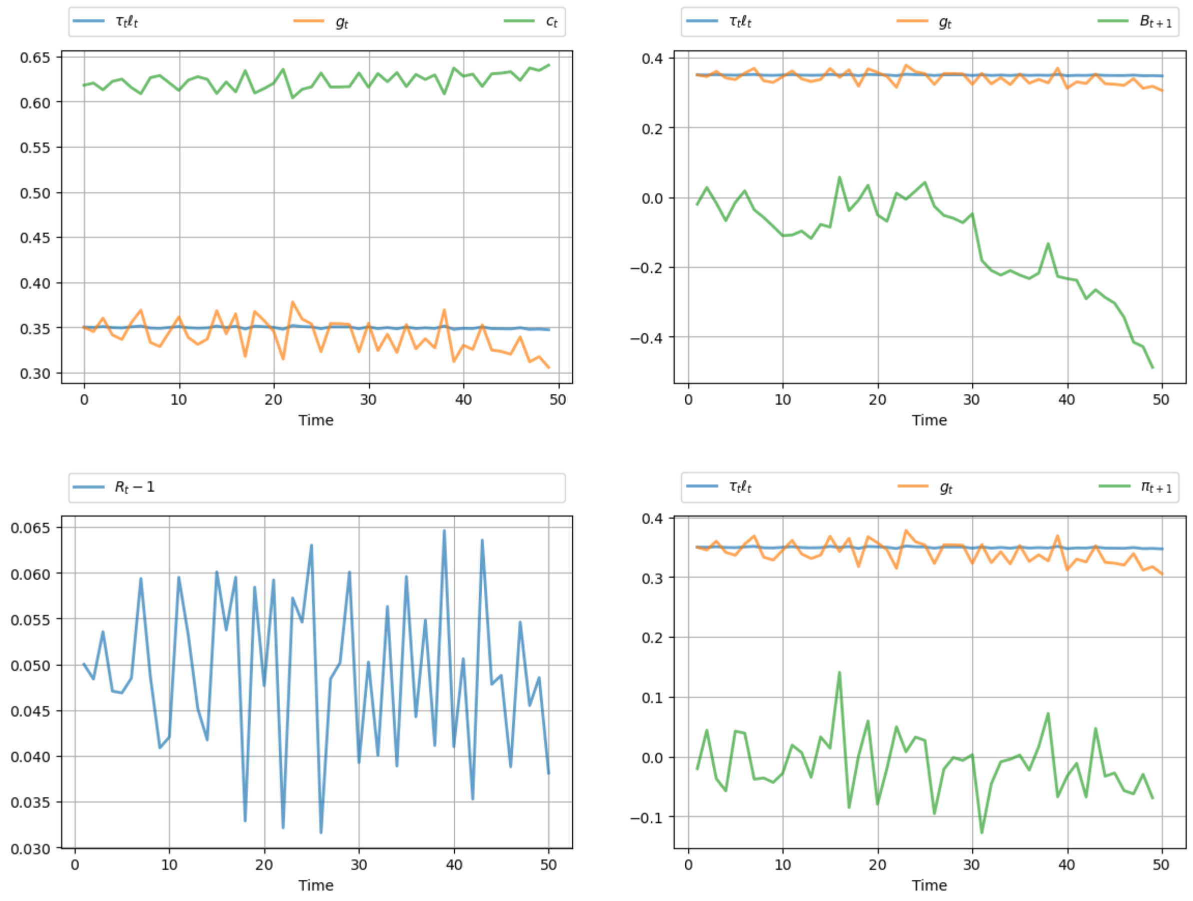
$$g_{t+1} - \mu_g = \rho(g_{t-3} - \mu_g) + C_g w_{g,t+1}$$

with $\rho = 0.95$ and $C_g = 0.7\sqrt{1-\rho^2}$.

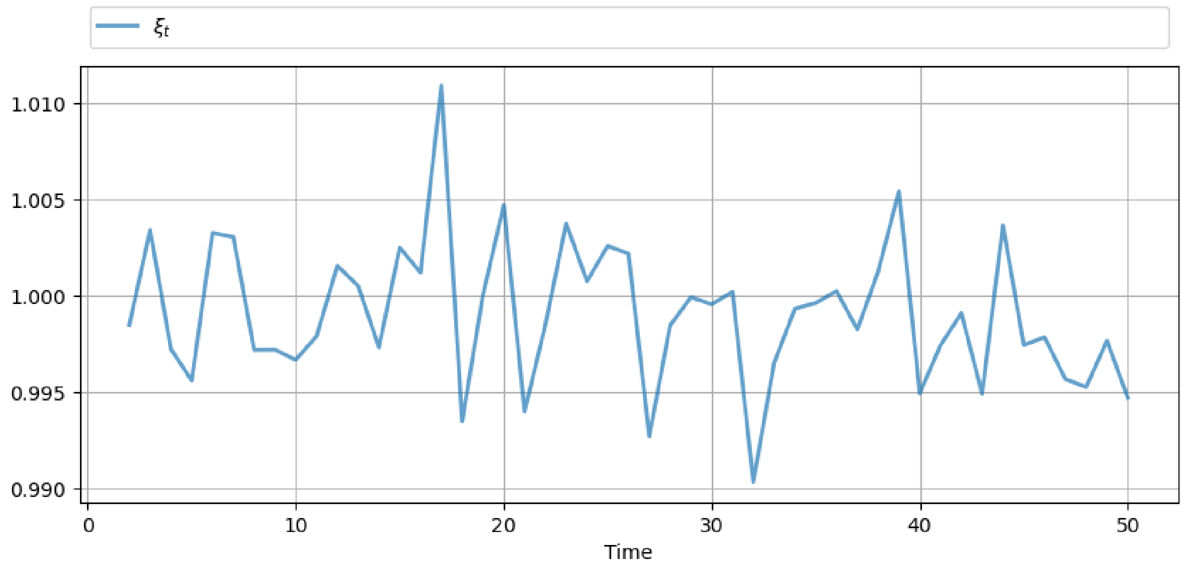
Produce the corresponding figures.

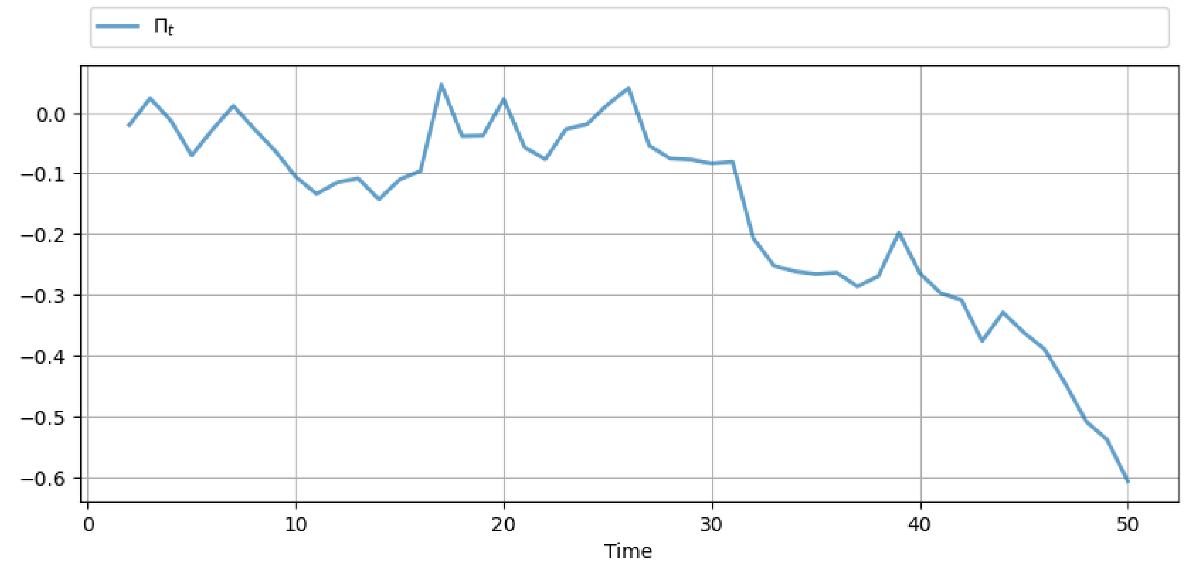
Solution to Exercise 17.5.1

```
# == Parameters == #
β = 1 / 1.05
ρ, mg = .95, .35
A = np.array([[0, 0, 0, ρ, mg*(1-ρ)],
              [1, 0, 0, 0,        0],
              [0, 1, 0, 0,        0],
              [0, 0, 1, 0,        0],
              [0, 0, 0, 0,        1]])
C = np.zeros((5, 1))
C[0, 0] = np.sqrt(1 - ρ**2) * mg / 8
Sg = np.array((1, 0, 0, 0, 0)).reshape(1, 5)
Sd = np.array((0, 0, 0, 0, 0)).reshape(1, 5)
# Chosen st. (Sc + Sg) * x0 = 1
Sb = np.array((0, 0, 0, 0, 2.135)).reshape(1, 5)
Ss = np.array((0, 0, 0, 0, 0)).reshape(1, 5)

economy = Economy(β=β, Sg=Sg, Sd=Sd, Sb=Sb,
                  Ss=Ss, discrete=False, proc=(A, C))

T = 50
path = compute_paths(T, economy)

gen_fig_1(path)
```



```
gen_fig_2(path)
```


ξ_t
1.010
1.005
1.000
0.995
0.990
0
10
20
30
40
50
Time


Π_t
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
0
10
20
30
40
50
Time

Part III

Dynamic Linear Economies

CHAPTER

EIGHTEEN

LINEAR STATE SPACE MODELS

Contents

"We may regard the present state of the universe as the effect of its past and the cause of its future" – Marquis de Laplace

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install quantecon
```

18.1 Overview

This lecture introduces the **linear state space** dynamic system.

The linear state space system is a generalization of the scalar AR(1) process we studied before.

This model is a workhorse that carries a powerful theory of prediction.

Its many applications include:

- representing dynamics of higher-order linear systems
- predicting the position of a system j steps into the future
- predicting a geometric sum of future values of a variable like
  - non-financial income
  - dividends on a stock

  - the money supply
  - a government deficit or surplus, etc.
- key ingredient of useful models
  - Friedman's permanent income model of consumption smoothing.
  - Barro's model of smoothing total tax collections.
  - Rational expectations version of Cagan's model of hyperinflation.
  - Sargent and Wallace's "unpleasant monetarist arithmetic," etc.

Let's start with some imports:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
from quantecon import LinearStateSpace
from scipy.stats import norm
import random
```

18.2 The Linear State Space Model

The objects in play are:

- An $n \times 1$ vector x_t denoting the **state** at time $t = 0, 1, 2, \ldots$.
- An IID sequence of $m \times 1$ random vectors $w_t \sim N(0, I)$.
- A $k \times 1$ vector y_t of **observations** at time $t = 0, 1, 2, \ldots$.
- An $n \times n$ matrix A called the **transition matrix**.
- An $n \times m$ matrix C called the **volatility matrix**.
- A $k \times n$ matrix G sometimes called the **output matrix**.

Here is the linear state-space system

$$
\begin{aligned}
x_{t+1} &= Ax_t + Cw_{t+1} \\
y_t &= Gx_t \\
x_0 &\sim N(\mu_0, \Sigma_0)
\end{aligned}
$$

18.2.1 Primitives

The primitives of the model are

1. the matrices A, C, G
2. shock distribution, which we have specialized to $N(0, I)$
3. the distribution of the initial condition x_0, which we have set to $N(\mu_0, \Sigma_0)$

Given A, C, G and draws of x_0 and $w_1, w_2, \ldots$, the model (18.1) pins down the values of the sequences $\{x_t\}$ and $\{y_t\}$.

Even without these draws, the primitives 1–3 pin down the *probability distributions* of $\{x_t\}$ and $\{y_t\}$.

Later we'll see how to compute these distributions and their moments.

Martingale Difference Shocks

We've made the common assumption that the shocks are independent standardized normal vectors.

But some of what we say will be valid under the assumption that $\{w_{t+1}\}$ is a **martingale difference sequence**.

A martingale difference sequence is a sequence that is zero mean when conditioned on past information.

In the present case, since $\{x_t\}$ is our state sequence, this means that it satisfies

$$\mathbb{E}[w_{t+1}|x_t, x_{t-1}, ...] = 0$$

This is a weaker condition than that $\{w_t\}$ is IID with $w_{t+1} \sim N(0, I)$.

18.2.2 Examples

By appropriate choice of the primitives, a variety of dynamics can be represented in terms of the linear state space model.

The following examples help to highlight this point.

They also illustrate the wise dictum *finding the state is an art*.

Second-order Difference Equation

Let $\{y_t\}$ be a deterministic sequence that satisfies

$$y_{t+1} = \phi_0 + \phi_1 y_t + \phi_2 y_{t-1} \quad \text{s.t.} \quad y_0, y_{-1} \text{ given} \tag{18.1}$$

To map (18.1) into our state space system (18.1), we set

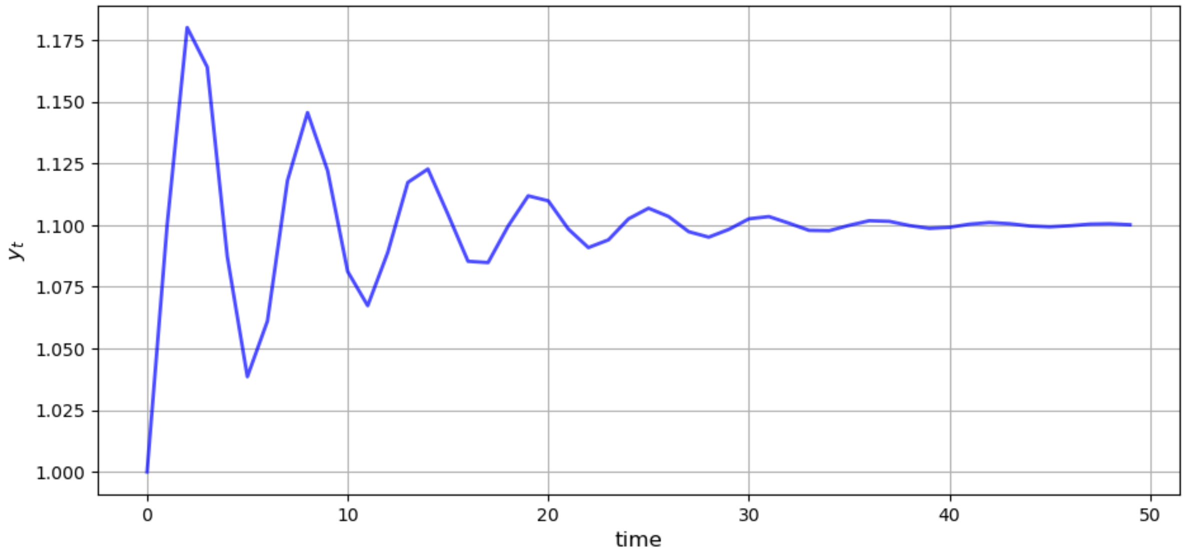
$$x_t = \begin{bmatrix} 1 \\ y_t \\ y_{t-1} \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 0 & 0 \\ \phi_0 & \phi_1 & \phi_2 \\ 0 & 1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad G = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

You can confirm that under these definitions, (18.1) and (18.1) agree.

The next figure shows the dynamics of this process when $\phi_0 = 1.1, \phi_1 = 0.8, \phi_2 = -0.8, y_0 = y_{-1} = 1$.

```python
def plot_lss(A,
         C,
         G,
         n=3,
         ts_length=50):

    ar = LinearStateSpace(A, C, G, mu_0=np.ones(n))
    x, y = ar.simulate(ts_length)

    fig, ax = plt.subplots()
    y = y.flatten()
    ax.plot(y, 'b-', lw=2, alpha=0.7)
    ax.grid()
    ax.set_xlabel('time', fontsize=12)
    ax.set_ylabel('$y_t$', fontsize=12)
    plt.show()
```

```
ϕ_0, ϕ_1, ϕ_2 = 1.1, 0.8, -0.8

A = [[1,     0,     0  ],
     [ϕ_0,   ϕ_1,   ϕ_2],
     [0,     1,     0  ]]

C = np.zeros((3, 1))
G = [0, 1, 0]

plot_lss(A, C, G)
```



Later you'll be asked to recreate this figure.

Univariate Autoregressive Processes

We can use (18.1) to represent the model

$$y_{t+1} = \phi_1 y_t + \phi_2 y_{t-1} + \phi_3 y_{t-2} + \phi_4 y_{t-3} + \sigma w_{t+1} \tag{18.2}$$

where $\{w_t\}$ is IID and standard normal.

To put this in the linear state space format we take $x_t = \begin{bmatrix} y_t & y_{t-1} & y_{t-2} & y_{t-3} \end{bmatrix}'$ and

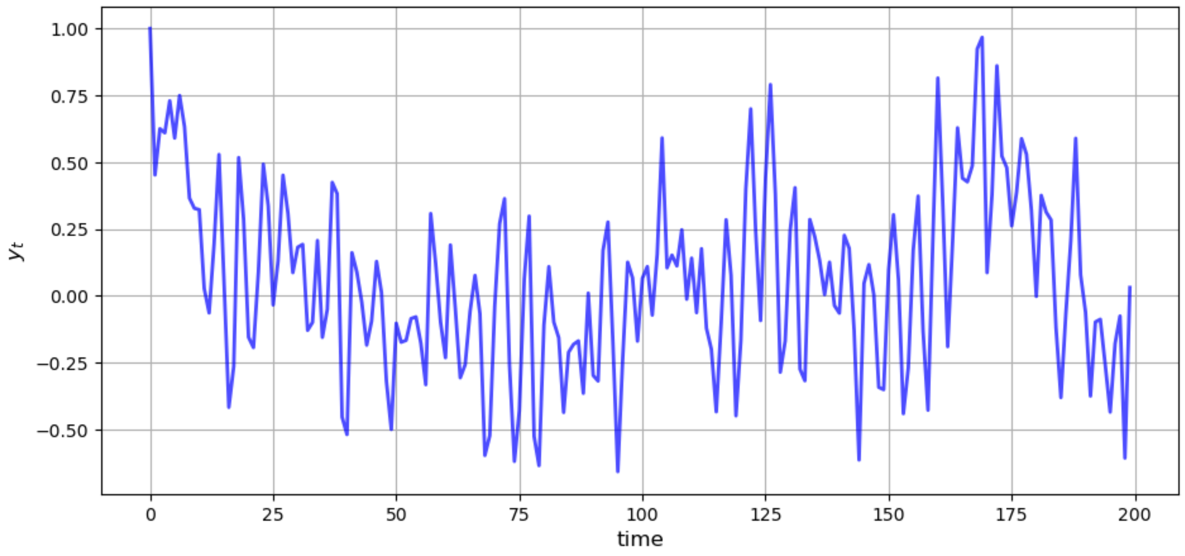
$$A = \begin{bmatrix} \phi_1 & \phi_2 & \phi_3 & \phi_4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} \sigma \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

The matrix A has the form of the *companion matrix* to the vector $\begin{bmatrix} \phi_1 & \phi_2 & \phi_3 & \phi_4 \end{bmatrix}$.

The next figure shows the dynamics of this process when

$$\phi_1 = 0.5, \phi_2 = -0.2, \phi_3 = 0, \phi_4 = 0.5, \sigma = 0.2, y_0 = y_{-1} = y_{-2} = y_{-3} = 1$$

```
ϕ_1, ϕ_2, ϕ_3, ϕ_4 = 0.5, -0.2, 0, 0.5
σ = 0.2

A_1 = [[ϕ_1,    ϕ_2,    ϕ_3,    ϕ_4],
       [1,      0,      0,      0  ],
       [0,      1,      0,      0  ],
       [0,      0,      1,      0  ]]

C_1 = [[σ],
       [0],
       [0],
       [0]]

G_1 = [1, 0, 0, 0]

plot_lss(A_1, C_1, G_1, n=4, ts_length=200)
```



Vector Autoregressions

Now suppose that

- y_t is a $k \times 1$ vector
- ϕ_j is a $k \times k$ matrix and
- w_t is $k \times 1$

Then (18.2) is termed a *vector autoregression*.

To map this into (18.1), we set

$$x_t = \begin{bmatrix} y_t \\ y_{t-1} \\ y_{t-2} \\ y_{t-3} \end{bmatrix} \quad A = \begin{bmatrix} \phi_1 & \phi_2 & \phi_3 & \phi_4 \\ I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \end{bmatrix} \quad C = \begin{bmatrix} \sigma \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad G = \begin{bmatrix} I & 0 & 0 & 0 \end{bmatrix}$$

where I is the $k \times k$ identity matrix and σ is a $k \times k$ matrix.

Seasonals

We can use (18.1) to represent

1. the *deterministic seasonal* $y_t = y_{t-4}$
2. the *indeterministic seasonal* $y_t = \phi_4 y_{t-4} + w_t$

In fact, both are special cases of (18.2).

With the deterministic seasonal, the transition matrix becomes

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

It is easy to check that $A^4 = I$, which implies that x_t is strictly periodic with period 4:[1]

$$x_{t+4} = x_t$$

Such an x_t process can be used to model deterministic seasonals in quarterly time series.

The *indeterministic* seasonal produces recurrent, but aperiodic, seasonal fluctuations.

Time Trends

The model $y_t = at + b$ is known as a *linear time trend*.

We can represent this model in the linear state space form by taking

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad G = \begin{bmatrix} a & b \end{bmatrix} \tag{18.3}$$

and starting at initial condition $x_0 = \begin{bmatrix} 0 & 1 \end{bmatrix}'$.

In fact, it's possible to use the state-space system to represent polynomial trends of any order.

For instance, we can represent the model $y_t = at^2 + bt + c$ in the linear state space form by taking

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad G = \begin{bmatrix} 2a & a + b & c \end{bmatrix}$$

and starting at initial condition $x_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}'$.

It follows that

$$A^t = \begin{bmatrix} 1 & t & t(t-1)/2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix}$$

Then $x_t' = \begin{bmatrix} t(t-1)/2 & t & 1 \end{bmatrix}$. You can now confirm that $y_t = G x_t$ has the correct form.

[1] The eigenvalues of A are $(1, -1, i, -i)$.

18.2.3 Moving Average Representations

A nonrecursive expression for x_t as a function of $x_0, w_1, w_2, \dots, w_t$ can be found by using (18.1) repeatedly to obtain

$$\begin{aligned} x_t &= Ax_{t-1} + Cw_t \\ &= A^2 x_{t-2} + ACw_{t-1} + Cw_t \\ &\quad \vdots \\ &= \sum_{j=0}^{t-1} A^j C w_{t-j} + A^t x_0 \end{aligned}$$

Representation (18.4) is a *moving average* representation.

It expresses $\{x_t\}$ as a linear function of

1. current and past values of the process $\{w_t\}$ and
2. the initial condition x_0

As an example of a moving average representation, let the model be

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

You will be able to show that $A^t = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}$ and $A^j C = \begin{bmatrix} 1 & 0 \end{bmatrix}'$.

Substituting into the moving average representation (18.4), we obtain

$$x_{1t} = \sum_{j=0}^{t-1} w_{t-j} + \begin{bmatrix} 1 & t \end{bmatrix} x_0$$

where x_{1t} is the first entry of x_t.

The first term on the right is a cumulated sum of martingale differences and is therefore a martingale.

The second term is a translated linear function of time.

For this reason, x_{1t} is called a *martingale with drift*.

18.3 Distributions and Moments

18.3.1 Unconditional Moments

Using (18.1), it's easy to obtain expressions for the (unconditional) means of x_t and y_t.

We'll explain what *unconditional* and *conditional* mean soon.

Letting $\mu_t := \mathbb{E}[x_t]$ and using linearity of expectations, we find that

$$\mu_{t+1} = A\mu_t \quad \text{with} \quad \mu_0 \text{ given} \tag{18.4}$$

Here μ_0 is a primitive given in (18.1).

The variance-covariance matrix of x_t is $\Sigma_t := \mathbb{E}[(x_t - \mu_t)(x_t - \mu_t)']$.

Using $x_{t+1} - \mu_{t+1} = A(x_t - \mu_t) + Cw_{t+1}$, we can determine this matrix recursively via

$$\Sigma_{t+1} = A\Sigma_t A' + CC' \quad \text{with} \quad \Sigma_0 \text{ given} \tag{18.5}$$

As with μ_0, the matrix Σ_0 is a primitive given in (18.1).

As a matter of terminology, we will sometimes call

- μ_t the *unconditional mean* of x_t
- Σ_t the *unconditional variance-covariance matrix* of x_t

This is to distinguish μ_t and Σ_t from related objects that use conditioning information, to be defined below.

However, you should be aware that these "unconditional" moments do depend on the initial distribution $N(\mu_0, \Sigma_0)$.

Moments of the Observables

Using linearity of expectations again we have

$$\mathbb{E}[y_t] = \mathbb{E}[Gx_t] = G\mu_t \tag{18.6}$$

The variance-covariance matrix of y_t is easily shown to be

$$\text{Var}[y_t] = \text{Var}[Gx_t] = G\Sigma_t G' \tag{18.7}$$

18.3.2 Distributions

In general, knowing the mean and variance-covariance matrix of a random vector is not quite as good as knowing the full distribution.

However, there are some situations where these moments alone tell us all we need to know.

These are situations in which the mean vector and covariance matrix are all of the **parameters** that pin down the population distribution.

One such situation is when the vector in question is Gaussian (i.e., normally distributed).

This is the case here, given

1. our Gaussian assumptions on the primitives
2. the fact that normality is preserved under linear operations

In fact, it's well-known that

$$u \sim N(\bar u, S) \quad \text{and} \quad v = a + Bu \implies v \sim N(a + B\bar u, BSB') \tag{18.8}$$

In particular, given our Gaussian assumptions on the primitives and the linearity of (18.1) we can see immediately that both x_t and y_t are Gaussian for all $t \geq 0$[2].

Since x_t is Gaussian, to find the distribution, all we need to do is find its mean and variance-covariance matrix.

But in fact we've already done this, in (18.4) and (18.5).

Letting μ_t and Σ_t be as defined by these equations, we have

$$x_t \sim N(\mu_t, \Sigma_t) \tag{18.9}$$

By similar reasoning combined with (18.6) and (18.7),

$$y_t \sim N(G\mu_t, G\Sigma_t G') \tag{18.10}$$

[2] The correct way to argue this is by induction. Suppose that x_t is Gaussian. Then (18.1) and (18.8) imply that x_{t+1} is Gaussian. Since x_0 is assumed to be Gaussian, it follows that every x_t is Gaussian. Evidently, this implies that each y_t is Gaussian.

18.3.3 Ensemble Interpretations

How should we interpret the distributions defined by (18.9)–(18.10)?

Intuitively, the probabilities in a distribution correspond to relative frequencies in a large population drawn from that distribution.

Let's apply this idea to our setting, focusing on the distribution of y_T for fixed T.

We can generate independent draws of y_T by repeatedly simulating the evolution of the system up to time T, using an independent set of shocks each time.

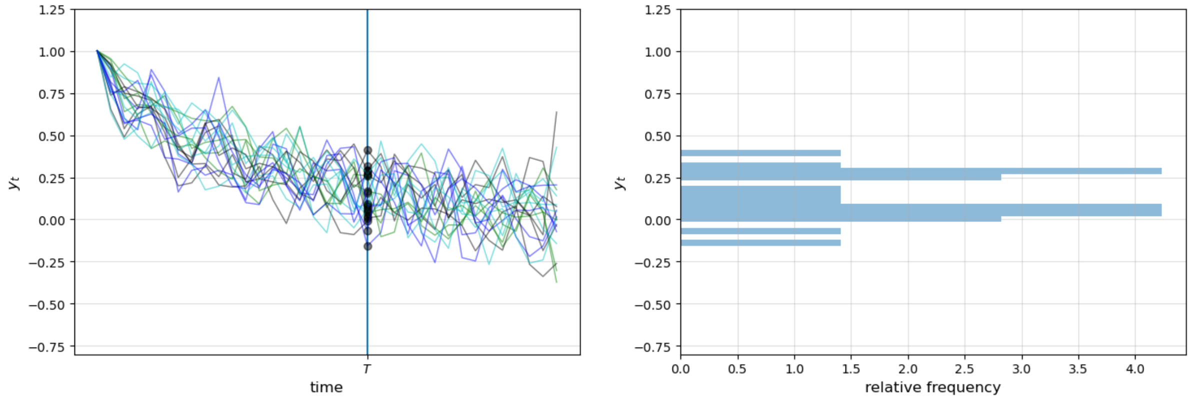
The next figure shows 20 simulations, producing 20 time series for $\{y_t\}$, and hence 20 draws of y_T.

The system in question is the univariate autoregressive model (18.2).

The values of y_T are represented by black dots in the left-hand figure

```python
def cross_section_plot(A,
                       C,
                       G,
                       T=20,                    # Set the time
                       ymin=-0.8,
                       ymax=1.25,
                       sample_size = 20,        # 20 observations/simulations
                       n=4):                    # The number of dimensions for the initial x0

    ar = LinearStateSpace(A, C, G, mu_0=np.ones(n))

    fig, axes = plt.subplots(1, 2, figsize=(16, 5))

    for ax in axes:
        ax.grid(alpha=0.4)
        ax.set_ylim(ymin, ymax)

    ax = axes[0]
    ax.set_ylim(ymin, ymax)
    ax.set_ylabel('$y_t$', fontsize=12)
    ax.set_xlabel('time', fontsize=12)
    ax.vlines((T,), -1.5, 1.5)

    ax.set_xticks((T,))
    ax.set_xticklabels(('$T$',))

    sample = []
    for i in range(sample_size):
        rcolor = random.choice(('c', 'g', 'b', 'k'))
        x, y = ar.simulate(ts_length=T+15)
        y = y.flatten()
        ax.plot(y, color=rcolor, lw=1, alpha=0.5)
        ax.plot((T,), (y[T],), 'ko', alpha=0.5)
        sample.append(y[T])

    y = y.flatten()
    axes[1].set_ylim(ymin, ymax)
    axes[1].set_ylabel('$y_t$', fontsize=12)
    axes[1].set_xlabel('relative frequency', fontsize=12)
    axes[1].hist(sample, bins=16, density=True, orientation='horizontal', alpha=0.5)
    plt.show()
```

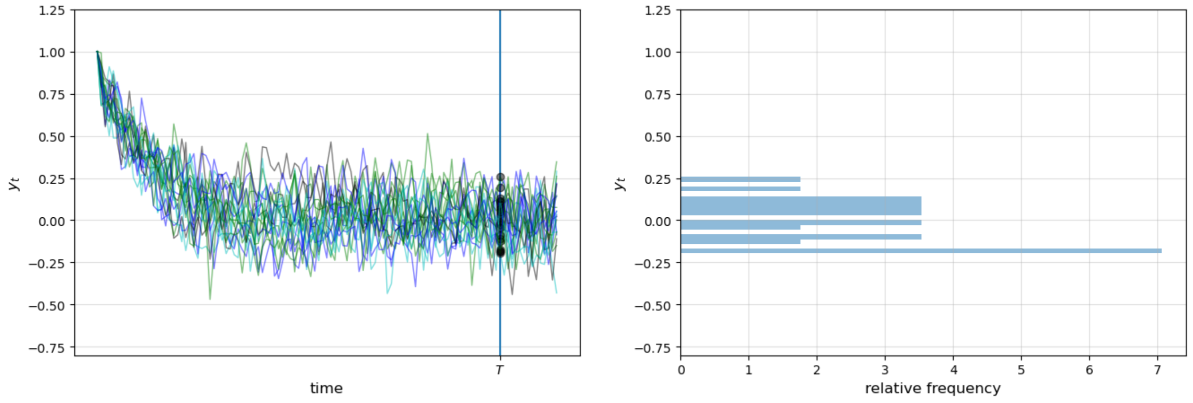
```
ϕ_1, ϕ_2, ϕ_3, ϕ_4 = 0.5, -0.2, 0, 0.5
σ = 0.1

A_2 = [[ϕ_1, ϕ_2, ϕ_3, ϕ_4],
       [1,     0,     0,     0],
       [0,     1,     0,     0],
       [0,     0,     1,     0]]

C_2 = [[σ], [0], [0], [0]]

G_2 = [1, 0, 0, 0]

cross_section_plot(A_2, C_2, G_2)
```



In the right-hand figure, these values are converted into a rotated histogram that shows relative frequencies from our sample of 20 y_T's.

Here is another figure, this time with 100 observations

```
t = 100
cross_section_plot(A_2, C_2, G_2, T=t)
```



Let's now try with 500,000 observations, showing only the histogram (without rotation)

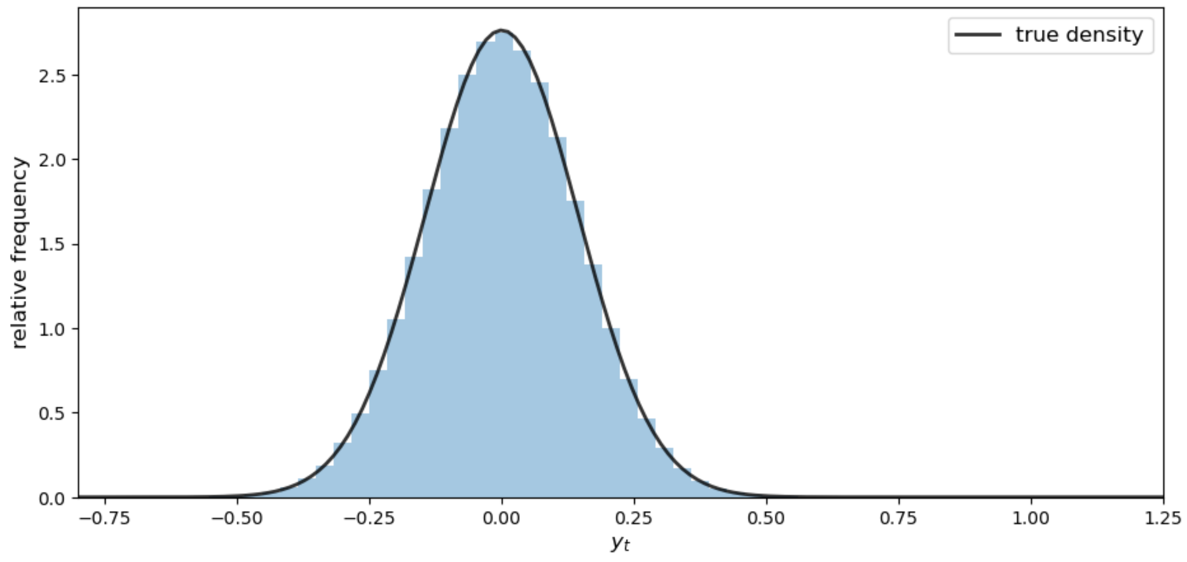
```
T = 100
ymin=-0.8
ymax=1.25
```

(continues on next page)

(continued from previous page)

```
sample_size = 500_000

ar = LinearStateSpace(A_2, C_2, G_2, mu_0=np.ones(4))
fig, ax = plt.subplots()
x, y = ar.simulate(sample_size)
mu_x, mu_y, Sigma_x, Sigma_y, Sigma_yx = ar.stationary_distributions()
f_y = norm(loc=float(mu_y), scale=float(np.sqrt(Sigma_y)))
y = y.flatten()
ygrid = np.linspace(ymin, ymax, 150)

ax.hist(y, bins=50, density=True, alpha=0.4)
ax.plot(ygrid, f_y.pdf(ygrid), 'k-', lw=2, alpha=0.8, label=r'true density')
ax.set_xlim(ymin, ymax)
ax.set_xlabel('$y_t$', fontsize=12)
ax.set_ylabel('relative frequency', fontsize=12)
ax.legend(fontsize=12)
plt.show()
```

```
/tmp/ipykernel_6430/1034809053.py:10: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  f_y = norm(loc=float(mu_y), scale=float(np.sqrt(Sigma_y)))
```



The black line is the population density of y_T calculated from (18.10).

The histogram and population distribution are close, as expected.

By looking at the figures and experimenting with parameters, you will gain a feel for how the population distribution depends on the model primitives *listed above*, as intermediated by the distribution's parameters.

Ensemble Means

In the preceding figure, we approximated the population distribution of y_T by

1. generating I sample paths (i.e., time series) where I is a large number
2. recording each observation y^i_T
3. histogramming this sample

Just as the histogram approximates the population distribution, the *ensemble* or *cross-sectional average*

$$\bar y_T := \frac{1}{I} \sum_{i=1}^I y_T^i$$

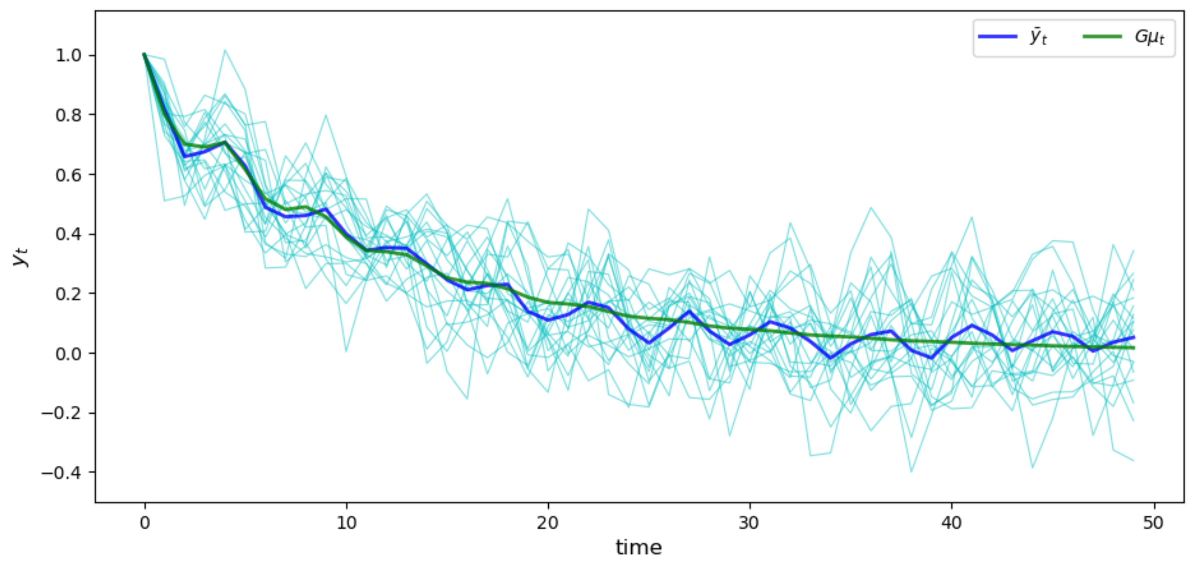
approximates the expectation $\mathbb{E}[y_T] = G \mu_T$ (as implied by the law of large numbers).

Here's a simulation comparing the ensemble averages and population means at time points $t = 0, \ldots, 50$.

The parameters are the same as for the preceding figures, and the sample size is relatively small ($I = 20$).

```
I = 20
T = 50
ymin = -0.5
ymax = 1.15

ar = LinearStateSpace(A_2, C_2, G_2, mu_0=np.ones(4))

fig, ax = plt.subplots()

ensemble_mean = np.zeros(T)
for i in range(I):
    x, y = ar.simulate(ts_length=T)
    y = y.flatten()
    ax.plot(y, 'c-', lw=0.8, alpha=0.5)
    ensemble_mean = ensemble_mean + y

ensemble_mean = ensemble_mean / I
ax.plot(ensemble_mean, color='b', lw=2, alpha=0.8, label='$\\bar y_t$')
m = ar.moment_sequence()

population_means = []
for t in range(T):
    μ_x, μ_y, Σ_x, Σ_y = next(m)
    population_means.append(float(μ_y))

ax.plot(population_means, color='g', lw=2, alpha=0.8, label='$G\mu_t$')
ax.set_ylim(ymin, ymax)
ax.set_xlabel('time', fontsize=12)
ax.set_ylabel('$y_t$', fontsize=12)
ax.legend(ncol=2)
plt.show()
```

```
/tmp/ipykernel_6430/3206934063.py:24: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  population_means.append(float(μ_y))
```



The ensemble mean for x_t is

$$\bar{x}_T := \frac{1}{I} \sum_{i=1}^{I} x_T^i \to \mu_T \qquad (I \to \infty)$$

The limit μ_T is a "long-run average".

(By *long-run average* we mean the average for an infinite ($I = \infty$) number of sample x_T's)

Another application of the law of large numbers assures us that

$$\frac{1}{I} \sum_{i=1}^{I} (x_T^i - \bar{x}_T)(x_T^i - \bar{x}_T)' \to \Sigma_T \qquad (I \to \infty)$$

18.3.4 Joint Distributions

In the preceding discussion, we looked at the distributions of x_t and y_t in isolation.

This gives us useful information but doesn't allow us to answer questions like

- what's the probability that $x_t \geq 0$ for all t?
- what's the probability that the process $\{y_t\}$ exceeds some value a before falling below b?
- etc., etc.

Such questions concern the *joint distributions* of these sequences.

To compute the joint distribution of $x_0, x_1, \ldots, x_T$, recall that joint and conditional densities are linked by the rule

$$p(x, y) = p(y \mid x) p(x) \qquad (\text{joint } = \text{ conditional } \times \text{ marginal})$$

From this rule we get $p(x_0, x_1) = p(x_1 \mid x_0) p(x_0)$.

The Markov property $p(x_t \mid x_{t-1}, \ldots, x_0) = p(x_t \mid x_{t-1})$ and repeated applications of the preceding rule lead us to

$$p(x_0, x_1, \ldots, x_T) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1} \mid x_t)$$

The marginal $p(x_0)$ is just the primitive $N(\mu_0, \Sigma_0)$.

In view of (18.1), the conditional densities are

$$p(x_{t+1} \,|\, x_t) = N(Ax_t, CC')$$

Autocovariance Functions

An important object related to the joint distribution is the *autocovariance function*

$$\Sigma_{t+j,t} := \mathbb{E}[(x_{t+j} - \mu_{t+j})(x_t - \mu_t)'] \tag{18.11}$$

Elementary calculations show that

$$\Sigma_{t+j,t} = A^j \Sigma_t \tag{18.12}$$

Notice that $\Sigma_{t+j,t}$ in general depends on both j, the gap between the two dates, and t, the earlier date.

18.4 Stationarity and Ergodicity

Stationarity and ergodicity are two properties that, when they hold, greatly aid analysis of linear state space models.

Let's start with the intuition.

18.4.1 Visualizing Stability

Let's look at some more time series from the same model that we analyzed above.

This picture shows cross-sectional distributions for y at times T, T', T''

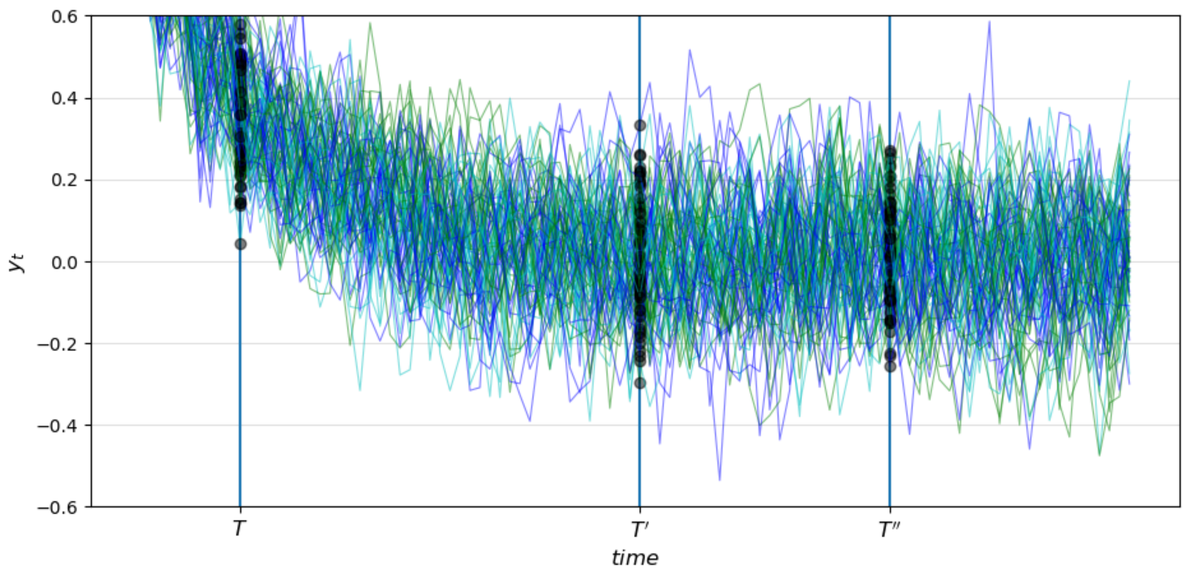
```
def cross_plot(A,
            C,
            G,
            steady_state='False',
            T0 = 10,
            T1 = 50,
            T2 = 75,
            T4 = 100):

    ar = LinearStateSpace(A, C, G, mu_0=np.ones(4))

    if steady_state == 'True':
        μ_x, μ_y, Σ_x, Σ_y, Σ_yx = ar.stationary_distributions()
        ar_state = LinearStateSpace(A, C, G, mu_0=μ_x, Sigma_0=Σ_x)

    ymin, ymax = -0.6, 0.6
    fig, ax = plt.subplots()
    ax.grid(alpha=0.4)
    ax.set_ylim(ymin, ymax)
    ax.set_ylabel('$y_t$', fontsize=12)
    ax.set_xlabel('$time$', fontsize=12)

    ax.vlines((T0, T1, T2), -1.5, 1.5)
```

(continues on next page)

(continued from previous page)

```
    ax.set_xticks((T0, T1, T2))
    ax.set_xticklabels(("$T$", "$T'$", "$T''$"), fontsize=12)
    for i in range(80):
        rcolor = random.choice(('c', 'g', 'b'))

        if steady_state == 'True':
            x, y = ar_state.simulate(ts_length=T4)
        else:
            x, y = ar.simulate(ts_length=T4)

        y = y.flatten()
        ax.plot(y, color=rcolor, lw=0.8, alpha=0.5)
        ax.plot((T0, T1, T2), (y[T0], y[T1], y[T2],), 'ko', alpha=0.5)
    plt.show()
```

```
cross_plot(A_2, C_2, G_2)
```



Note how the time series "settle down" in the sense that the distributions at T' and T'' are relatively similar to each other — but unlike the distribution at T.

Apparently, the distributions of y_t converge to a fixed long-run distribution as $t \to \infty$.

When such a distribution exists it is called a *stationary distribution*.

18.4.2 Stationary Distributions

In our setting, a distribution ψ_∞ is said to be *stationary* for x_t if

$$x_t \sim \psi_\infty \quad \text{and} \quad x_{t+1} = Ax_t + Cw_{t+1} \quad \implies \quad x_{t+1} \sim \psi_\infty$$

Since

1. in the present case, all distributions are Gaussian
2. a Gaussian distribution is pinned down by its mean and variance-covariance matrix

we can restate the definition as follows: ψ_∞ is stationary for x_t if

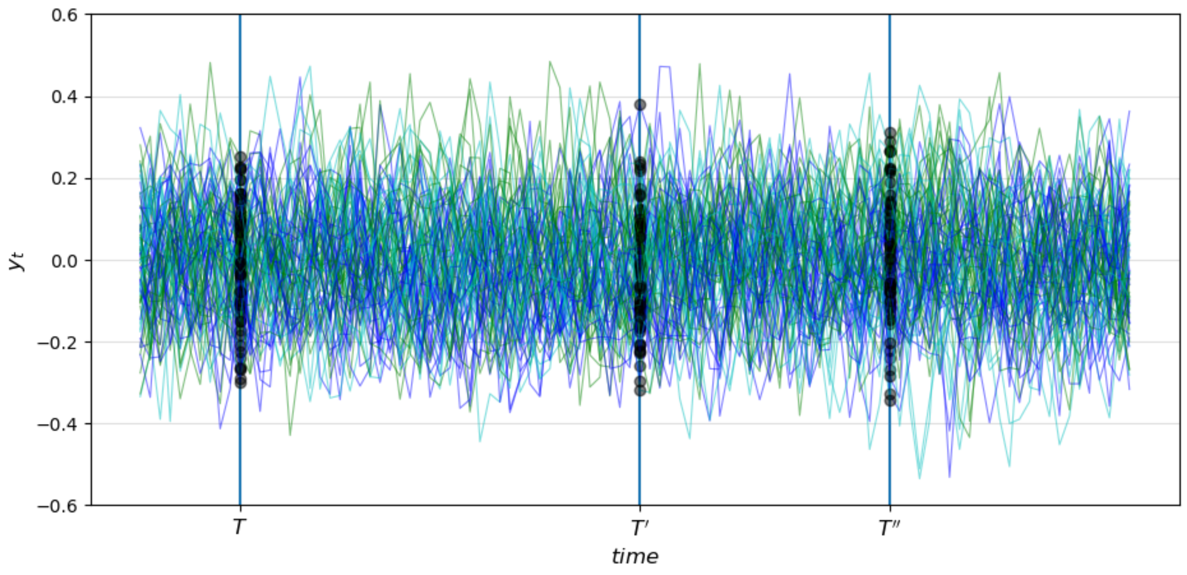
$$\psi_\infty = N(\mu_\infty, \Sigma_\infty)$$

where μ_∞ and Σ_∞ are fixed points of (18.4) and (18.5) respectively.

18.4.3 Covariance Stationary Processes

Let's see what happens to the preceding figure if we start x_0 at the stationary distribution.

```
cross_plot(A_2, C_2, G_2, steady_state='True')
```



Now the differences in the observed distributions at T, T' and T'' come entirely from random fluctuations due to the finite sample size.

By

- our choosing $x_0 \sim N(\mu_\infty, \Sigma_\infty)$
- the definitions of μ_∞ and Σ_∞ as fixed points of (18.4) and (18.5) respectively

we've ensured that

$$\mu_t = \mu_\infty \quad \text{and} \quad \Sigma_t = \Sigma_\infty \quad \text{for all } t$$

Moreover, in view of (18.12), the autocovariance function takes the form $\Sigma_{t+j,t} = A^j \Sigma_\infty$, which depends on j but not on t.

This motivates the following definition.

A process $\{x_t\}$ is said to be *covariance stationary* if

- both μ_t and Σ_t are constant in t
- $\Sigma_{t+j,t}$ depends on the time gap j but not on time t

In our setting, $\{x_t\}$ will be covariance stationary if μ_0, Σ_0, A, C assume values that imply that none of $\mu_t, \Sigma_t, \Sigma_{t+j,t}$ depends on t.

18.4.4 Conditions for Stationarity

The Globally Stable Case

The difference equation $\mu_{t+1} = A\mu_t$ is known to have *unique* fixed point $\mu_\infty = 0$ if all eigenvalues of A have moduli strictly less than unity.

That is, if `(np.absolute(np.linalg.eigvals(A)) < 1).all() == True`.

The difference equation (18.5) also has a unique fixed point in this case, and, moreover

$$\mu_t \to \mu_\infty = 0 \quad \text{and} \quad \Sigma_t \to \Sigma_\infty \quad \text{as} \quad t \to \infty$$

regardless of the initial conditions μ_0 and Σ_0.

This is the *globally stable case* — see these notes for more a theoretical treatment.

However, global stability is more than we need for stationary solutions, and often more than we want.

To illustrate, consider *our second order difference equation example*.

Here the state is $x_t = \begin{bmatrix} 1 & y_t & y_{t-1} \end{bmatrix}'$.

Because of the constant first component in the state vector, we will never have $\mu_t \to 0$.

How can we find stationary solutions that respect a constant state component?

Processes with a Constant State Component

To investigate such a process, suppose that A and C take the form

$$A = \begin{bmatrix} A_1 & a \\ 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} C_1 \\ 0 \end{bmatrix}$$

where

- A_1 is an $(n-1) \times (n-1)$ matrix
- a is an $(n-1) \times 1$ column vector

Let $x_t = \begin{bmatrix} x_{1t}' & 1 \end{bmatrix}'$ where x_{1t} is $(n-1) \times 1$.

It follows that

$$x_{1,t+1} = A_1 x_{1t} + a + C_1 w_{t+1}$$

Let $\mu_{1t} = \mathbb{E}[x_{1t}]$ and take expectations on both sides of this expression to get

$$\mu_{1,t+1} = A_1 \mu_{1,t} + a \tag{18.13}$$

Assume now that the moduli of the eigenvalues of A_1 are all strictly less than one.

Then (18.13) has a unique stationary solution, namely,

$$\mu_{1\infty} = (I - A_1)^{-1} a$$

The stationary value of μ_t itself is then $\mu_\infty := \begin{bmatrix} \mu_{1\infty}' & 1 \end{bmatrix}'$.

The stationary values of Σ_t and $\Sigma_{t+j,t}$ satisfy

$$\begin{aligned} \Sigma_\infty &= A \Sigma_\infty A' + CC' \\ \Sigma_{t+j,t} &= A^j \Sigma_\infty \end{aligned}$$

Notice that here $\Sigma_{t+j,t}$ depends on the time gap j but not on calendar time t.

In conclusion, if

- $x_0 \sim N(\mu_\infty, \Sigma_\infty)$ and
- the moduli of the eigenvalues of A_1 are all strictly less than unity

then the $\{x_t\}$ process is covariance stationary, with constant state component.

Note: If the eigenvalues of A_1 are less than unity in modulus, then (a) starting from any initial value, the mean and variance-covariance matrix both converge to their stationary values; and (b) iterations on (18.5) converge to the fixed point of the *discrete Lyapunov equation* in the first line of (18.14).

18.4.5 Ergodicity

Let's suppose that we're working with a covariance stationary process.

In this case, we know that the ensemble mean will converge to μ_∞ as the sample size I approaches infinity.

Averages over Time

Ensemble averages across simulations are interesting theoretically, but in real life, we usually observe only a *single* realization $\{x_t, y_t\}_{t=0}^T$.

So now let's take a single realization and form the time-series averages

$$\bar{x} := \frac{1}{T}\sum_{t=1}^T x_t \quad \text{and} \quad \bar{y} := \frac{1}{T}\sum_{t=1}^T y_t$$

Do these time series averages converge to something interpretable in terms of our basic state-space representation?

The answer depends on something called *ergodicity*.

Ergodicity is the property that time series and ensemble averages coincide.

More formally, ergodicity implies that time series sample averages converge to their expectation under the stationary distribution.

In particular,

- $\frac{1}{T}\sum_{t=1}^T x_t \to \mu_\infty$
- $\frac{1}{T}\sum_{t=1}^T (x_t - \bar{x}_T)(x_t - \bar{x}_T)' \to \Sigma_\infty$
- $\frac{1}{T}\sum_{t=1}^T (x_{t+j} - \bar{x}_T)(x_t - \bar{x}_T)' \to A^j\Sigma_\infty$

In our linear Gaussian setting, any covariance stationary process is also ergodic.

18.5 Noisy Observations

In some settings, the observation equation $y_t = Gx_t$ is modified to include an error term.

Often this error term represents the idea that the true state can only be observed imperfectly.

To include an error term in the observation we introduce

- An IID sequence of $\ell \times 1$ random vectors $v_t \sim N(0, I)$.
- A $k \times \ell$ matrix H.

and extend the linear state-space system to

$$\begin{aligned} x_{t+1} &= Ax_t + Cw_{t+1} \\ y_t &= Gx_t + Hv_t \\ x_0 &\sim N(\mu_0, \Sigma_0) \end{aligned}$$

The sequence $\{v_t\}$ is assumed to be independent of $\{w_t\}$.

The process $\{x_t\}$ is not modified by noise in the observation equation and its moments, distributions and stability properties remain the same.

The unconditional moments of y_t from (18.6) and (18.7) now become

$$\mathbb{E}[y_t] = \mathbb{E}[Gx_t + Hv_t] = G\mu_t \tag{18.14}$$

The variance-covariance matrix of y_t is easily shown to be

$$\text{Var}[y_t] = \text{Var}[Gx_t + Hv_t] = G\Sigma_t G' + HH' \tag{18.15}$$

The distribution of y_t is therefore

$$y_t \sim N(G\mu_t, G\Sigma_t G' + HH')$$

18.6 Prediction

The theory of prediction for linear state space systems is elegant and simple.

18.6.1 Forecasting Formulas – Conditional Means

The natural way to predict variables is to use conditional distributions.

For example, the optimal forecast of x_{t+1} given information known at time t is

$$\mathbb{E}_t[x_{t+1}] := \mathbb{E}[x_{t+1} \mid x_t, x_{t-1}, \dots, x_0] = Ax_t$$

The right-hand side follows from $x_{t+1} = Ax_t + Cw_{t+1}$ and the fact that w_{t+1} is zero mean and independent of $x_t, x_{t-1}, \dots, x_0$.

That $\mathbb{E}_t[x_{t+1}] = \mathbb{E}[x_{t+1} \mid x_t]$ is an implication of $\{x_t\}$ having the *Markov property*.

The one-step-ahead forecast error is

$$x_{t+1} - \mathbb{E}_t[x_{t+1}] = Cw_{t+1}$$

The covariance matrix of the forecast error is

$$\mathbb{E}[(x_{t+1} - \mathbb{E}_t[x_{t+1}])(x_{t+1} - \mathbb{E}_t[x_{t+1}])'] = CC'$$

More generally, we'd like to compute the j-step ahead forecasts $\mathbb{E}_t[x_{t+j}]$ and $\mathbb{E}_t[y_{t+j}]$.

With a bit of algebra, we obtain

$$x_{t+j} = A^j x_t + A^{j-1} C w_{t+1} + A^{j-2} C w_{t+2} + \cdots + A^0 C w_{t+j}$$

In view of the IID property, current and past state values provide no information about future values of the shock.

Hence $\mathbb{E}_t[w_{t+k}] = \mathbb{E}[w_{t+k}] = 0$.

It now follows from linearity of expectations that the j-step ahead forecast of x is

$$\mathbb{E}_t[x_{t+j}] = A^j x_t$$

The j-step ahead forecast of y is therefore

$$\mathbb{E}_t[y_{t+j}] = \mathbb{E}_t[G x_{t+j} + H v_{t+j}] = G A^j x_t$$

18.6.2 Covariance of Prediction Errors

It is useful to obtain the covariance matrix of the vector of j-step-ahead prediction errors

$$x_{t+j} - \mathbb{E}_t[x_{t+j}] = \sum_{s=0}^{j-1} A^s C w_{t-s+j} \tag{18.16}$$

Evidently,

$$V_j := \mathbb{E}_t[(x_{t+j} - \mathbb{E}_t[x_{t+j}])(x_{t+j} - \mathbb{E}_t[x_{t+j}])'] = \sum_{k=0}^{j-1} A^k C C' A^{k'} \tag{18.17}$$

V_j defined in (18.17) can be calculated recursively via $V_1 = CC'$ and

$$V_j = CC' + A V_{j-1} A', \quad j \geq 2 \tag{18.18}$$

V_j is the *conditional covariance matrix* of the errors in forecasting x_{t+j}, conditioned on time t information x_t.

Under particular conditions, V_j converges to

$$V_\infty = CC' + A V_\infty A' \tag{18.19}$$

Equation (18.19) is an example of a *discrete Lyapunov* equation in the covariance matrix V_∞.

A sufficient condition for V_j to converge is that the eigenvalues of A be strictly less than one in modulus.

Weaker sufficient conditions for convergence associate eigenvalues equaling or exceeding one in modulus with elements of C that equal 0.

18.7 Code

Our preceding simulations and calculations are based on code in the file lss.py from the QuantEcon.py package.

The code implements a class for handling linear state space models (simulations, calculating moments, etc.).

One Python construct you might not be familiar with is the use of a generator function in the method `moment_sequence()`.

Go back and read the relevant documentation if you've forgotten how generator functions work.

Examples of usage are given in the solutions to the exercises.

18.8 Exercises

Exercise 18.8.1

In several contexts, we want to compute forecasts of geometric sums of future random variables governed by the linear state-space system (18.1).

We want the following objects

- Forecast of a geometric sum of future x's, or $\mathbb{E}_t \left[\sum_{j=0}^{\infty} \beta^j x_{t+j}\right]$.
- Forecast of a geometric sum of future y's, or $\mathbb{E}_t \left[\sum_{j=0}^{\infty} \beta^j y_{t+j}\right]$.

These objects are important components of some famous and interesting dynamic models.

For example,

- if $\{y_t\}$ is a stream of dividends, then $\mathbb{E}\left[\sum_{j=0}^{\infty} \beta^j y_{t+j} | x_t\right]$ is a model of a stock price
- if $\{y_t\}$ is the money supply, then $\mathbb{E}\left[\sum_{j=0}^{\infty} \beta^j y_{t+j} | x_t\right]$ is a model of the price level

Show that:

$$\mathbb{E}_t \left[\sum_{j=0}^{\infty} \beta^j x_{t+j}\right] = [I - \beta A]^{-1} x_t$$

and

$$\mathbb{E}_t \left[\sum_{j=0}^{\infty} \beta^j y_{t+j}\right] = G[I - \beta A]^{-1} x_t$$

what must the modulus for every eigenvalue of A be less than?

Solution to Exercise 18.8.1

Suppose that every eigenvalue of A has modulus strictly less than $\frac{1}{\beta}$.

It then follows that $I + \beta A + \beta^2 A^2 + \cdots = [I - \beta A]^{-1}$.

This leads to our formulas:

- Forecast of a geometric sum of future x's

$$\mathbb{E}_t \left[\sum_{j=0}^{\infty} \beta^j x_{t+j}\right] = [I + \beta A + \beta^2 A^2 + \cdots] x_t = [I - \beta A]^{-1} x_t$$

- Forecast of a geometric sum of future y's

$$\mathbb{E}_t \left[\sum_{j=0}^{\infty} \beta^j y_{t+j} \right] = G[I + \beta A + \beta^2 A^2 + \cdots] x_t = G[I - \beta A]^{-1} x_t$$

CHAPTER

NINETEEN

RECURSIVE MODELS OF DYNAMIC LINEAR ECONOMIES

Contents

"Mathematics is the art of giving the same name to different things" – Henri Poincare

"Complete market economies are all alike" – Robert E. Lucas, Jr., (1989)

"Every partial equilibrium model can be reinterpreted as a general equilibrium model." – Anonymous

19.1 A Suite of Models

This lecture presents a class of linear-quadratic-Gaussian models of general economic equilibrium designed by Lars Peter Hansen and Thomas J. Sargent [Hansen and Sargent, 2013].

The class of models is implemented in a Python class DLE that is part of quantecon.

Subsequent lectures use the DLE class to implement various instances that have appeared in the economics literature

1. *Growth in Dynamic Linear Economies*
2. *Lucas Asset Pricing using DLE*

3. *IRFs in Hall Model*
4. *Permanent Income Using the DLE class*
5. *Rosen schooling model*
6. *Cattle cycles*
7. *Shock Non Invertibility*

19.1.1 Overview of the Models

In saying that "complete markets are all alike", Robert E. Lucas, Jr. was noting that all of them have

- a commodity space.
- a space dual to the commodity space in which prices reside.
- endowments of resources.
- peoples' preferences over goods.
- physical technologies for transforming resources into goods.
- random processes that govern shocks to technologies and preferences and associated information flows.
- a single budget constraint per person.
- the existence of a representative consumer even when there are many people in the model.
- a concept of competitive equilibrium.
- theorems connecting competitive equilibrium allocations to allocations that would be chosen by a benevolent social planner.

The models have **no frictions** such as …

- Enforcement difficulties
- Information asymmetries
- Other forms of transactions costs
- Externalities

The models extensively use the powerful ideas of

- Indexing commodities and their prices by time (John R. Hicks).
- Indexing commodities and their prices by chance (Kenneth Arrow).

Much of the imperialism of complete markets models comes from applying these two tricks.

The Hicks trick of indexing commodities by time is the idea that **dynamics are a special case of statics**.

The Arrow trick of indexing commodities by chance is the idea that **analysis of trade under uncertainty is a special case of the analysis of trade under certainty**.

The [Hansen and Sargent, 2013] class of models specify the commodity space, preferences, technologies, stochastic shocks and information flows in ways that allow the models to be analyzed completely using only the tools of linear time series models and linear-quadratic optimal control described in the two lectures Linear State Space Models and Linear Quadratic Control.

There are costs and benefits associated with the simplifications and specializations needed to make a particular model fit within the [Hansen and Sargent, 2013] class

- the costs are that linear-quadratic structures are sometimes too confining.

- benefits include computational speed, simplicity, and ability to analyze many model features analytically or nearly analytically.

A variety of superficially different models are all instances of the [Hansen and Sargent, 2013] class of models

- Lucas asset pricing model
- Lucas-Prescott model of investment under uncertainty
- Asset pricing models with habit persistence
- Rosen-Topel equilibrium model of housing
- Rosen schooling models
- Rosen-Murphy-Scheinkman model of cattle cycles
- Hansen-Sargent-Tallarini model of robustness and asset pricing
- Many more …

The diversity of these models conceals an essential unity that illustrates the quotation by Robert E. Lucas, Jr., with which we began this lecture.

19.1.2 Forecasting?

A consequence of a single budget constraint per person plus the Hicks-Arrow tricks is that households and firms need not forecast.

But there exist equivalent structures called **recursive competitive equilibria** in which they do appear to need to forecast.

In these structures, to forecast, households and firms use:

- equilibrium pricing functions, and
- knowledge of the Markov structure of the economy's state vector.

19.1.3 Theory and Econometrics

For an application of the [Hansen and Sargent, 2013] class of models, the outcome of theorizing is a stochastic process, i.e., a probability distribution over sequences of prices and quantities, indexed by parameters describing preferences, technologies, and information flows.

Another name for that object is a likelihood function, a key object of both frequentist and Bayesian statistics.

There are two important uses of an **equilibrium stochastic process** or **likelihood function**.

The first is to solve the **direct problem**.

The **direct problem** takes as inputs values of the parameters that define preferences, technologies, and information flows and as an output characterizes or simulates random paths of quantities and prices.

The second use of an equilibrium stochastic process or likelihood function is to solve the **inverse problem**.

The **inverse problem** takes as an input a time series sample of observations on a subset of prices and quantities determined by the model and from them makes inferences about the parameters that define the model's preferences, technologies, and information flows.

19.1.4 More Details

A [Hansen and Sargent, 2013] economy consists of **lists of matrices** that describe peoples' household technologies, their preferences over consumption services, their production technologies, and their information sets.

There are complete markets in history-contingent commodities.

Competitive equilibrium allocations and prices

- satisfy equations that are easy to write down and solve
- have representations that are convenient econometrically

Different example economies manifest themselves simply as different settings for various matrices.

[Hansen and Sargent, 2013] use these tools:

- A theory of recursive dynamic competitive economies
- Linear optimal control theory
- Recursive methods for estimating and interpreting vector autoregressions

The models are flexible enough to express alternative senses of a representative household

- A single 'stand-in' household of the type used to good effect by Edward C. Prescott.
- Heterogeneous households satisfying conditions for Gorman aggregation into a representative household.
- Heterogeneous household technologies that violate conditions for Gorman aggregation but are still susceptible to aggregation into a single representative household via 'non-Gorman' or 'mongrel' aggregation'.

These three alternative types of aggregation have different consequences in terms of how prices and allocations can be computed.

In particular, can prices and an aggregate allocation be computed before the equilibrium allocation to individual heterogeneous households is computed?

- Answers are "Yes" for Gorman aggregation, "No" for non-Gorman aggregation.

In summary, the insights and practical benefits from economics to be introduced in this lecture are

- Deeper understandings that come from recognizing common underlying structures.
- Speed and ease of computation that comes from unleashing a common suite of Python programs.

We'll use the following **mathematical tools**

- Stochastic Difference Equations (Linear).
- Duality: LQ Dynamic Programming and Linear Filtering are the same things mathematically.
- The Spectral Factorization Identity (for understanding vector autoregressions and non-Gorman aggregation).

So here is our roadmap.

We'll describe sets of matrices that pin down

- Information
- Technologies
- Preferences

Then we'll describe

- Equilibrium concept and computation
- Econometric representation and estimation

19.1.5 Stochastic Model of Information Flows and Outcomes

We'll use stochastic linear difference equations to describe information flows **and** equilibrium outcomes.

The sequence $\{w_t : t = 1, 2, \ldots\}$ is said to be a martingale difference sequence adapted to $\{J_t : t = 0, 1, \ldots\}$ if $E(w_{t+1}|J_t) = 0$ for $t = 0, 1, \ldots$.

The sequence $\{w_t : t = 1, 2, \ldots\}$ is said to be conditionally homoskedastic if $E(w_{t+1}w'_{t+1} \mid J_t) = I$ for $t = 0, 1, \ldots$.

We assume that the $\{w_t : t = 1, 2, \ldots\}$ process is conditionally homoskedastic.

Let $\{x_t : t = 1, 2, \ldots\}$ be a sequence of n-dimensional random vectors, i.e. an n-dimensional stochastic process.

The process $\{x_t : t = 1, 2, \ldots\}$ is constructed recursively using an initial random vector $x_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_0)$ and a time-invariant law of motion:

$$x_{t+1} = Ax_t + Cw_{t+1}$$

for $t = 0, 1, \ldots$ where A is an n by n matrix and C is an n by N matrix.

Evidently, the distribution of x_{t+1} conditional on x_t is $\mathcal{N}(Ax_t, CC')$.

19.1.6 Information Sets

Let J_0 be generated by x_0 and J_t be generated by $x_0, w_1, \ldots, w_t$, which means that J_t consists of the set of all measurable functions of $\{x_0, w_1, \ldots, w_t\}$.

19.1.7 Prediction Theory

The optimal forecast of x_{t+1} given current information is

$$E(x_{t+1} \mid J_t) = Ax_t$$

and the one-step-ahead forecast error is

$$x_{t+1} - E(x_{t+1} \mid J_t) = Cw_{t+1}$$

The covariance matrix of x_{t+1} conditioned on J_t is

$$E(x_{t+1} - E(x_{t+1} \mid J_t))(x_{t+1} - E(x_{t+1} \mid J_t))' = CC'$$

A nonrecursive expression for x_t as a function of $x_0, w_1, w_2, \ldots, w_t$ is

$$\begin{aligned} x_t &= Ax_{t-1} + Cw_t \\ &= A^2 x_{t-2} + ACw_{t-1} + Cw_t \\ &= \left[\sum_{\tau=0}^{t-1} A^\tau C w_{t-\tau}\right] + A^t x_0 \end{aligned}$$

Shift forward in time:

$$x_{t+j} = \sum_{s=0}^{j-1} A^s C w_{t+j-s} + A^j x_t$$

Projecting on the information set $\{x_0, w_t, w_{t-1}, \ldots, w_1\}$ gives

$$E_t x_{t+j} = A^j x_t$$

where $E_t(\cdot) \equiv E[(\cdot) \mid x_0, w_t, w_{t-1}, \ldots, w_1] = E(\cdot) \mid J_t$, and x_t is in J_t.

It is useful to obtain the covariance matrix of the j-step-ahead prediction error $x_{t+j} - E_t x_{t+j} = \sum_{s=0}^{j-1} A^s C w_{t-s+j}$.

Evidently,

$$E_t(x_{t+j} - E_t x_{t+j})(x_{t+j} - E_t x_{t+j})' = \sum_{k=0}^{j-1} A^k C C' A^{k'} \equiv v_j$$

v_j can be calculated recursively via

$$\begin{aligned} v_1 &= CC' \\ v_j &= CC' + A v_{j-1} A', \quad j \geq 2 \end{aligned}$$

19.1.8 Orthogonal Decomposition

To decompose these covariances into parts attributable to the individual components of w_t, we let i_τ be an N-dimensional column vector of zeroes except in position τ, where there is a one. Define a matrix $v_{j,\tau}$

$$v_{j,\tau} = \sum_{k=0}^{j-1} A^k C i_\tau i_\tau' C' A'^k.$$

Note that $\sum_{\tau=1}^N i_\tau i_\tau' = I$, so that we have

$$\sum_{\tau=1}^N v_{j,\tau} = v_j$$

Evidently, the matrices $\{v_{j,\tau}, \tau = 1, \ldots, N\}$ give an orthogonal decomposition of the covariance matrix of j-step-ahead prediction errors into the parts attributable to each of the components $\tau = 1, \ldots, N$.

19.1.9 Taste and Technology Shocks

$E(w_t \mid J_{t-1}) = 0$ and $E(w_t w_t' \mid J_{t-1}) = I$ for $t = 1, 2, \ldots$

$$b_t = U_b z_t \text{ and } d_t = U_d z_t,$$

U_b and U_d are matrices that select entries of z_t. The law of motion for $\{z_t : t = 0, 1, \ldots\}$ is

$$z_{t+1} = A_{22} z_t + C_2 w_{t+1} \text{ for } t = 0, 1, \ldots$$

where z_0 is a given initial condition. The eigenvalues of the matrix A_{22} have absolute values that are less than or equal to one.

Thus, in summary, our model of **information and shocks** is

$$\begin{aligned} z_{t+1} &= A_{22} z_t + C_2 w_{t+1} \\ b_t &= U_b z_t \\ d_t &= U_d z_t. \end{aligned}$$

We can now briefly summarize other components of our economies, in particular

- Production technologies
- Household technologies
- Household preferences

19.1.10 Production Technology

Where c_t is a vector of consumption rates, k_t is a vector of physical capital goods, g_t is a vector intermediate productions goods, d_t is a vector of technology shocks, the production technology is

$$\begin{aligned}\Phi_c c_t + \Phi_g g_t + \Phi_i i_t &= \Gamma k_{t-1} + d_t \\ k_t &= \Delta_k k_{t-1} + \Theta_k i_t \\ g_t \cdot g_t &= \ell_t^2\end{aligned}$$

Here $\Phi_c, \Phi_g, \Phi_i, \Gamma, \Delta_k, \Theta_k$ are all matrices conformable to the vectors they multiply and ℓ_t is a disutility generating resource supplied by the household.

For technical reasons that facilitate computations, we make the following.

Assumption: $[\Phi_c \ \Phi_g]$ is nonsingular.

19.1.11 Household Technology

Households confront a technology that allows them to devote consumption goods to construct a vector h_t of household capital goods and a vector s_t of utility generating house services

$$\begin{aligned}s_t &= \Lambda h_{t-1} + \Pi c_t \\ h_t &= \Delta_h h_{t-1} + \Theta_h c_t\end{aligned}$$

where $\Lambda, \Pi, \Delta_h, \Theta_h$ are matrices that pin down the household technology.

We make the following

Assumption: The absolute values of the eigenvalues of Δ_h are less than or equal to one.

Below, we'll outline further assumptions that we shall occasionally impose.

19.1.12 Preferences

Where b_t is a stochastic process of preference shocks that will play the role of demand shifters, the representative household orders stochastic processes of consumption services s_t according to

$$\left(\frac{1}{2}\right) E \sum_{t=0}^{\infty} \beta^t [(s_t - b_t) \cdot (s_t - b_t) + \ell_t^2] \big| J_0, \ 0 < \beta < 1$$

We now proceed to give examples of production and household technologies that appear in various models that appear in the literature.

First, we give examples of production Technologies

$$\Phi_c c_t + \Phi_g g_t + \Phi_i i_t = \Gamma k_{t-1} + d_t$$

$$\mid g_t \mid \leq \ell_t$$

so we'll be looking for specifications of the matrices $\Phi_c, \Phi_g, \Phi_i, \Gamma, \Delta_k, \Theta_k$ that define them.

19.1.13 Endowment Economy

There is a single consumption good that cannot be stored over time.

In time period t, there is an endowment d_t of this single good.

There is neither a capital stock, nor an intermediate good, nor a rate of investment.

So $c_t = d_t$.

To implement this specification, we can choose A_{22}, C_2, and U_d to make d_t follow any of a variety of stochastic processes.

To satisfy our earlier rank assumption, we set:

$$c_t + i_t = d_{1t}$$

$$g_t = \phi_1 i_t$$

where ϕ_1 is a small positive number.

To implement this version, we set $\Delta_k = \Theta_k = 0$ and

$$\Phi_c = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \Phi_i = \begin{bmatrix} 1 \\ \phi_1 \end{bmatrix}, \ \Phi_g = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \ \Gamma = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ d_t = \begin{bmatrix} d_{1t} \\ 0 \end{bmatrix}$$

We can use this specification to create a linear-quadratic version of Lucas's (1978) asset pricing model.

19.1.14 Single-Period Adjustment Costs

There is a single consumption good, a single intermediate good, and a single investment good.

The technology is described by

$$\begin{aligned} c_t &= \gamma k_{t-1} + d_{1t}, \ \ \gamma > 0 \\ \phi_1 i_t &= g_t + d_{2t}, \ \ \phi_1 > 0 \\ \ell_t^2 &= g_t^2 \\ k_t &= \delta_k k_{t-1} + i_t, \ 0 < \delta_k < 1 \end{aligned}$$

Set

$$\Phi_c = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ \Phi_g = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \ \Phi_i = \begin{bmatrix} 0 \\ \phi_1 \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} \gamma \\ 0 \end{bmatrix}, \ \Delta_k = \delta_k, \ \Theta_k = 1$$

We set A_{22}, C_2 and U_d to make $(d_{1t}, d_{2t})' = d_t$ follow a desired stochastic process.

Now we describe some examples of preferences, which as we have seen are ordered by

$$-\left(\frac{1}{2}\right) E \sum_{t=0}^{\infty} \beta^t \left[(s_t - b_t) \cdot (s_t - b_t) + (\ell_t)^2\right] \mid J_0 \quad , \ 0 < \beta < 1$$

where household services are produced via the household technology

$$h_t = \Delta_h h_{t-1} + \Theta_h c_t$$

$$s_t = \Lambda h_{t-1} + \Pi c_t$$

and we make

Assumption: The absolute values of the eigenvalues of Δ_h are less than or equal to one.

Later we shall introduce **canonical** household technologies that satisfy an 'invertibility' requirement relating sequences $\{s_t\}$ of services and $\{c_t\}$ of consumption flows.

And we'll describe how to obtain a canonical representation of a household technology from one that is not canonical.

Here are some examples of household preferences.

Time Separable preferences

$$-\frac{1}{2}E\sum_{t=0}^{\infty}\beta^t\left[(c_t-b_t)^2+\ell_t^2\right] \mid J_0 \quad , 0<\beta<1$$

Consumer Durables

$$h_t=\delta_h h_{t-1}+c_t \quad , 0<\delta_h<1$$

Services at t are related to the stock of durables at the beginning of the period:

$$s_t=\lambda h_{t-1} \ , \ \lambda>0$$

Preferences are ordered by

$$-\frac{1}{2}E\sum_{t=0}^{\infty}\beta^t\left[(\lambda h_{t-1}-b_t)^2+\ell_t^2\right] \mid J_0$$

Set $\Delta_h=\delta_h, \Theta_h=1, \Lambda=\lambda, \Pi=0$.

Habit Persistence

$$-\Big(\frac{1}{2}\Big)\,E\sum_{t=0}^{\infty}\beta^t\Big[\big(c_t-\lambda(1-\delta_h)\sum_{j=0}^{\infty}\,\delta_h^j\,c_{t-j-1}-b_t\big)^2+\ell_t^2\Big]\big|J_0$$

$$0<\beta<1 \ , \ 0<\delta_h<1 \ , \ \lambda>0$$

Here the effective bliss point $b_t+\lambda(1-\delta_h)\sum_{j=0}^{\infty}\delta_h^j\,c_{t-j-1}$ shifts in response to a moving average of past consumption.

Initial Conditions

Preferences of this form require an initial condition for the geometric sum $\sum_{j=0}^{\infty}\delta_h^j c_{t-j-1}$ that we specify as an initial condition for the 'stock of household durables,' h_{-1}.

Set

$$h_t=\delta_h h_{t-1}+(1-\delta_h)c_t \quad , 0<\delta_h<1$$

$$h_t=(1-\delta_h)\sum_{j=0}^{t}\delta_h^j\,c_{t-j}+\delta_h^{t+1}\,h_{-1}$$

$$s_t=-\lambda h_{t-1}+c_t, \ \lambda>0$$

To implement, set $\Lambda=-\lambda, \ \Pi=1, \ \Delta_h=\delta_h, \ \Theta_h=1-\delta_h$.

Seasonal Habit Persistence

$$-\Big(\frac{1}{2}\Big)\,E\sum_{t=0}^{\infty}\beta^t\Big[\big(c_t-\lambda(1-\delta_h)\sum_{j=0}^{\infty}\,\delta_h^j\,c_{t-4j-4}-b_t\big)^2+\ell_t^2\Big]$$

$$0<\beta<1 \ , \ 0<\delta_h<1 \ , \ \lambda>0$$

Here the effective bliss point $b_t + \lambda(1-\delta_h)\sum_{j=0}^\infty \delta_h^j \, c_{t-4j-4}$ shifts in response to a moving average of past consumptions of the same quarter.

To implement, set

$$\tilde{h}_t = \delta_h \tilde{h}_{t-4} + (1-\delta_h) c_t \quad , \ 0 < \delta_h < 1$$

This implies that

$$h_t = \begin{bmatrix} \tilde{h}_t \\ \tilde{h}_{t-1} \\ \tilde{h}_{t-2} \\ \tilde{h}_{t-3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \delta_h \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \tilde{h}_{t-1} \\ \tilde{h}_{t-2} \\ \tilde{h}_{t-3} \\ \tilde{h}_{t-4} \end{bmatrix} + \begin{bmatrix} (1-\delta_h) \\ 0 \\ 0 \\ 0 \end{bmatrix} c_t$$

with consumption services

$$s_t = -\begin{bmatrix} 0 & 0 & 0 & -\lambda \end{bmatrix} h_{t-1} + c_t \quad , \ \lambda > 0$$

Adjustment Costs.

Recall

$$-\Big(\frac{1}{2}\Big) E \sum_{t=0}^\infty \beta^t [(c_t - b_{1t})^2 + \lambda^2 (c_t - c_{t-1})^2 + \ell_t^2] \mid J_0$$

$$0 < \beta < 1 \quad , \ \lambda > 0$$

To capture adjustment costs, set

$$h_t = c_t$$

$$s_t = \begin{bmatrix} 0 \\ -\lambda \end{bmatrix} h_{t-1} + \begin{bmatrix} 1 \\ \lambda \end{bmatrix} c_t$$

so that

$$s_{1t} = c_t$$

$$s_{2t} = \lambda (c_t - c_{t-1})$$

We set the first component b_{1t} of b_t to capture the stochastic bliss process and set the second component identically equal to zero.

Thus, we set $\Delta_h = 0, \Theta_h = 1$

$$\Lambda = \begin{bmatrix} 0 \\ -\lambda \end{bmatrix} \ , \ \Pi = \begin{bmatrix} 1 \\ \lambda \end{bmatrix}$$

Multiple Consumption Goods

$$\Lambda = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \ \text{ and } \ \Pi = \begin{bmatrix} \pi_1 & 0 \\ \pi_2 & \pi_3 \end{bmatrix}$$

$$-\frac{1}{2}\beta^t (\Pi c_t - b_t)'(\Pi c_t - b_t)$$

$$\mu_t = -\beta^t [\Pi' \Pi \, c_t - \Pi' \, b_t]$$

$$c_t = -(\Pi'\Pi)^{-1} \beta^{-t} \mu_t + (\Pi'\Pi)^{-1} \Pi' b_t$$

This is called the **Frisch demand function** for consumption.

We can think of the vector μ_t as playing the role of prices, up to a common factor, for all dates and states.

The scale factor is determined by the choice of numeraire.

Notions of **substitutes and complements** can be defined in terms of these Frisch demand functions.

Two goods can be said to be **substitutes** if the cross-price effect is positive and to be **complements** if this effect is negative.

Hence this classification is determined by the off-diagonal element of $-(\Pi'\Pi)^{-1}$, which is equal to $\pi_2\pi_3/\det(\Pi'\Pi)$.

If π_2 and π_3 have the same sign, the goods are substitutes.

If they have opposite signs, the goods are complements.

To summarize, our economic structure consists of the matrices that define the following components:

Information and shocks

$$\begin{aligned} z_{t+1} &= A_{22}z_t + C_2 w_{t+1} \\ b_t &= U_b z_t \\ d_t &= U_d z_t \end{aligned}$$

Production Technology

$$\begin{aligned} \Phi_c c_t + \Phi_g g_t + \Phi_i i_t &= \Gamma k_{t-1} + d_t \\ k_t &= \Delta_k k_{t-1} + \Theta_k i_t \\ g_t \cdot g_t &= \ell_t^2 \end{aligned}$$

Household Technology

$$\begin{aligned} s_t &= \Lambda h_{t-1} + \Pi c_t \\ h_t &= \Delta_h h_{t-1} + \Theta_h c_t \end{aligned}$$

Preferences

$$\Big(\frac{1}{2}\Big) E \sum_{t=0}^{\infty} \beta^t [(s_t - b_t)\cdot(s_t - b_t) + \ell_t^2] \big| J_0, \ 0 < \beta < 1$$

Next steps: we move on to discuss two closely connected concepts

- A Planning Problem or Optimal Resource Allocation Problem
- Competitive Equilibrium

19.1.15 Optimal Resource Allocation

Imagine a planner who chooses sequences $\{c_t, i_t, g_t\}_{t=0}^{\infty}$ to maximize

$$-(1/2) E \sum_{t=0}^{\infty} \beta^t [(s_t - b_t)\cdot(s_t - b_t) + g_t \cdot g_t] \big| J_0$$

subject to the constraints

$$\begin{aligned} \Phi_c c_t + \Phi_g g_t + \Phi_i i_t &= \Gamma k_{t-1} + d_t, \\ k_t &= \Delta_k k_{t-1} + \Theta_k i_t, \\ h_t &= \Delta_h h_{t-1} + \Theta_h c_t, \\ s_t &= \Lambda h_{t-1} + \Pi c_t, \\ z_{t+1} &= A_{22} z_t + C_2 w_{t+1}, \ b_t = U_b z_t, \ \text{ and } \ d_t = U_d z_t \end{aligned}$$

and initial conditions for h_{-1}, k_{-1}, and z_0.

Throughout, we shall impose the following **square summability** conditions

$$E\sum_{t=0}^{\infty}\beta^t h_t \cdot h_t \mid J_0 < \infty \text{ and } E\sum_{t=0}^{\infty}\beta^t k_t \cdot k_t \mid J_0 < \infty$$

Define:

$$L_0^2 = [\{y_t\} : y_t \text{ is a random variable in } J_t \text{ and } E\sum_{t=0}^{\infty}\beta^t y_t^2 \mid J_0 < +\infty]$$

Thus, we require that each component of h_t and each component of k_t belong to L_0^2.

We shall compare and utilize two approaches to solving the planning problem

- Lagrangian formulation
- Dynamic programming

19.1.16 Lagrangian Formulation

Form the Lagrangian

$$\begin{aligned}\mathcal{L} = -E\sum_{t=0}^{\infty}\beta^t\Bigg[&\left(\frac{1}{2}\right)[(s_t - b_t)\cdot(s_t - b_t) + g_t \cdot g_t] \\ &+ M_t^{d\prime}\cdot(\Phi_c c_t + \Phi_g g_t + \Phi_i i_t - \Gamma k_{t-1} - d_t) \\ &+ M_t^{k\prime}\cdot(k_t - \Delta_k k_{t-1} - \Theta_k i_t) \\ &+ M_t^{h\prime}\cdot(h_t - \Delta_h h_{t-1} - \Theta_h c_t) \\ &+ M_t^{s\prime}\cdot(s_t - \Lambda h_{t-1} - \Pi c_t)\Bigg]\Bigg| J_0\end{aligned}$$

The planner maximizes $\mathcal{L}$ with respect to the quantities $\{c_t, i_t, g_t\}_{t=0}^{\infty}$ and minimizes with respect to the Lagrange multipliers $M_t^d, M_t^k, M_t^h, M_t^s$.

First-order necessary conditions for maximization with respect to c_t, g_t, h_t, i_t, k_t, and s_t, respectively, are:

$$\begin{aligned}-\Phi_c' M_t^d + \Theta_h' M_t^h + \Pi' M_t^s &= 0, \\ -g_t - \Phi_g' M_t^d &= 0, \\ -M_t^h + \beta E(\Delta_h' M_{t+1}^h + \Lambda' M_{t+1}^s) \mid J_t &= 0, \\ -\Phi_i' M_t^d + \Theta_k' M_t^k &= 0, \\ -M_t^k + \beta E(\Delta_k' M_{t+1}^k + \Gamma' M_{t+1}^d) \mid J_t &= 0, \\ -s_t + b_t - M_t^s &= 0\end{aligned}$$

for $t = 0, 1, \dots$.

In addition, we have the complementary slackness conditions (these recover the original transition equations) and also transversality conditions

$$\lim_{t\to\infty}\beta^t E[M_t^{k\prime} k_t] \mid J_0 = 0$$
$$\lim_{t\to\infty}\beta^t E[M_t^{h\prime} h_t] \mid J_0 = 0$$

The system formed by the FONCs and the transition equations can be handed over to Python.

Python will solve the planning problem for fixed parameter values.

Here are the **Python Ready Equations**

$$\begin{aligned}
-\Phi_c' M_t^d + \Theta_h' M_t^h + \Pi' M_t^s &= 0, \\
-g_t - \Phi_g' M_t^d &= 0, \\
-M_t^h + \beta E(\Delta_h' M_{t+1}^h + \Lambda' M_{t+1}^s) \mid J_t &= 0, \\
-\Phi_i' M_t^d + \Theta_k' M_t^k &= 0, \\
-M_t^k + \beta E(\Delta_k' M_{t+1}^k + \Gamma' M_{t+1}^d) \mid J_t &= 0, \\
-s_t + b_t - M_t^s &= 0 \\
\Phi_c c_t + \Phi_g g_t + \Phi_i i_t &= \Gamma k_{t-1} + d_t, \\
k_t &= \Delta_k k_{t-1} + \Theta_k i_t, \\
h_t &= \Delta_h h_{t-1} + \Theta_h c_t, \\
s_t &= \Lambda h_{t-1} + \Pi c_t, \\
z_{t+1} &= A_{22} z_t + C_2 w_{t+1}, \ b_t = U_b z_t, \ \text{and} \ d_t = U_d z_t
\end{aligned}$$

The Lagrange multipliers or **shadow prices** satisfy

$$M_t^s = b_t - s_t$$

$$M_t^h = E\left[\sum_{\tau=1}^{\infty} \beta^\tau (\Delta_h')^{\tau-1} \Lambda' M_{t+\tau}^s \mid J_t\right]$$

$$M_t^d = \begin{bmatrix} \Phi_c' \\ \Phi_g' \end{bmatrix}^{-1} \begin{bmatrix} \Theta_h' M_t^h + \Pi' M_t^s \\ -g_t \end{bmatrix}$$

$$M_t^k = E\left[\sum_{\tau=1}^{\infty} \beta^\tau (\Delta_k')^{\tau-1} \Gamma' M_{t+\tau}^d \mid J_t\right]$$

$$M_t^i = \Theta_k' M_t^k$$

Although it is possible to use matrix operator methods to solve the above **Python ready equations**, that is not the approach we'll use.

Instead, we'll use dynamic programming to get recursive representations for both quantities and shadow prices.

19.1.17 Dynamic Programming

Dynamic Programming always starts with the word **let**.

Thus, let $V(x_0)$ be the optimal value function for the planning problem as a function of the initial state vector x_0.

(Thus, in essence, dynamic programming amounts to an application of a **guess and verify** method in which we begin with a guess about the answer to the problem we want to solve. That's why we start with **let** $V(x_0)$ be the (value of the) answer to the problem, then establish and verify a bunch of conditions $V(x_0)$ has to satisfy if indeed it is the answer)

The optimal value function $V(x)$ satisfies the **Bellman equation**

$$V(x_0) = \max_{c_0, i_0, g_0} [-.5[(s_0 - b_0) \cdot (s_0 - b_0) + g_0 \cdot g_0] + \beta E V(x_1)]$$

subject to the linear constraints

$$\begin{aligned}
\Phi_c c_0 + \Phi_g g_0 + \Phi_i i_0 &= \Gamma k_{-1} + d_0, \\
k_0 &= \Delta_k k_{-1} + \Theta_k i_0, \\
h_0 &= \Delta_h h_{-1} + \Theta_h c_0, \\
s_0 &= \Lambda h_{-1} + \Pi c_0, \\
z_1 &= A_{22} z_0 + C_2 w_1, \ b_0 = U_b z_0 \ \text{and} \ d_0 = U_d z_0
\end{aligned}$$

Because this is a linear-quadratic dynamic programming problem, it turns out that the value function has the form

$$V(x) = x'Px + \rho$$

Thus, we want to solve an instance of the following linear-quadratic dynamic programming problem:

Choose a contingency plan for $\{x_{t+1}, u_t\}_{t=0}^\infty$ to maximize

$$-E\sum_{t=0}^\infty \beta^t[x_t'Rx_t + u_t'Qu_t + 2u_t'W'x_t],\ 0 < \beta < 1$$

subject to

$$x_{t+1} = Ax_t + Bu_t + Cw_{t+1},\ t \geq 0$$

where x_0 is given; x_t is an $n \times 1$ vector of state variables, and u_t is a $k \times 1$ vector of control variables.

We assume w_{t+1} is a martingale difference sequence with $Ew_tw_t' = I$, and that C is a matrix conformable to x and w.

The optimal value function $V(x)$ satisfies the Bellman equation

$$V(x_t) = \max_{u_t}\Big\{-(x_t'Rx_t + u_t'Qu_t + 2u_t'Wx_t) + \beta E_tV(x_{t+1})\Big\}$$

where maximization is subject to

$$x_{t+1} = Ax_t + Bu_t + Cw_{t+1},\ t \geq 0$$

$$V(x_t) = -x_t'Px_t - \rho$$

P satisfies

$$P = R + \beta A'PA - (\beta A'PB + W)(Q + \beta B'PB)^{-1}(\beta B'PA + W')$$

This equation in P is called the **algebraic matrix Riccati equation**.

The optimal decision rule is $u_t = -Fx_t$, where

$$F = (Q + \beta B'PB)^{-1}(\beta B'PA + W')$$

The optimum decision rule for u_t is independent of the parameters C, and so of the noise statistics.

Iterating on the Bellman operator leads to

$$V_{j+1}(x_t) = \max_{u_t}\Big\{-(x_t'Rx_t + u_t'Qu_t + 2u_t'Wx_t) + \beta E_tV_j(x_{t+1})\Big\}$$

$$V_j(x_t) = -x_t'P_jx_t - \rho_j$$

where P_j and ρ_j satisfy the equations

$$\begin{aligned} P_{j+1} &= R + \beta A'P_jA - (\beta A'P_jB + W)(Q + \beta B'P_jB)^{-1}(\beta B'P_jA + W') \\ \rho_{j+1} &= \beta\rho_j + \beta \text{ trace } P_jCC' \end{aligned}$$

We can now state the planning problem as a dynamic programming problem

$$\max_{\{u_t, x_{t+1}\}} -E\sum_{t=0}^\infty \beta^t[x_t'Rx_t + u_t'Qu_t + 2u_t'W'x_t], \quad 0 < \beta < 1$$

where maximization is subject to

$$x_{t+1} = Ax_t + Bu_t + Cw_{t+1},\ t \geq 0$$

$$x_t = \begin{bmatrix} h_{t-1} \\ k_{t-1} \\ z_t \end{bmatrix}, \qquad u_t = i_t$$

where

$$A = \begin{bmatrix} \Delta_h & \Theta_h U_c [\Phi_c \ \ \Phi_g]^{-1} \Gamma & \Theta_h U_c [\Phi_c \ \ \Phi_g]^{-1} U_d \\ 0 & \Delta_k & 0 \\ 0 & 0 & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} -\Theta_h U_c [\Phi_c \ \ \Phi_g]^{-1} \Phi_i \\ \Theta_k \\ 0 \end{bmatrix}, \ C = \begin{bmatrix} 0 \\ 0 \\ C_2 \end{bmatrix}$$

$$\begin{bmatrix} x_t \\ u_t \end{bmatrix}' S \begin{bmatrix} x_t \\ u_t \end{bmatrix} = \begin{bmatrix} x_t \\ u_t \end{bmatrix}' \begin{bmatrix} R & W \\ W' & Q \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}$$

$$S = (G'G + H'H)/2$$

$$H = [\Lambda \ \vdots \ \Pi U_c [\Phi_c \ \ \Phi_g]^{-1} \Gamma \ \vdots \ \Pi U_c [\Phi_c \ \ \Phi_g]^{-1} U_d - U_b \ \vdots \ -\Pi U_c [\Phi_c \ \ \Phi_g]^{-1} \Phi_i]$$

$$G = U_g [\Phi_c \ \ \Phi_g]^{-1} [0 \ \vdots \ \Gamma \ \vdots \ U_d \ \vdots \ -\Phi_i].$$

Lagrange multipliers as gradient of value function

A useful fact is that Lagrange multipliers equal gradients of the planner's value function

$$\begin{aligned} \mathcal{M}_t^k &= M_k x_t \ \text{ and } \ M_t^h = M_h x_t \ \text{ where} \\ M_k &= 2\beta[0 \ I \ 0] P A^o \\ M_h &= 2\beta[I \ 0 \ 0] P A^o \end{aligned}$$

$$\mathcal{M}_t^s = M_s x_t \ \text{ where } \ M_s = (S_b - S_s) \ \text{ and } \ S_b = [0 \ 0 \ U_b]$$

$$\mathcal{M}_t^d = M_d x_t \ \text{ where } \ M_d = \begin{bmatrix} \Phi_c' \\ \Phi_g' \end{bmatrix}^{-1} \begin{bmatrix} \Theta_h' M_h + \Pi' M_s \\ -S_g \end{bmatrix}$$

$$\mathcal{M}_t^c = M_c x_t \ \text{ where } \ M_c = \Theta_h' M_h + \Pi' M_s$$

$$\mathcal{M}_t^i = M_i x_t \ \text{ where } \ M_i = \Theta_k' M_k$$

We will use this fact and these equations to compute competitive equilibrium prices.

19.1.18 Other mathematical infrastructure

Let's start with describing the **commodity space** and **pricing functional** for our competitive equilibrium.

For the **commodity space**, we use

$$L_0^2 = [\{y_t\} : y_t \text{ is a random variable in } \ J_t \ \text{ and } \ E \sum_{t=0}^{\infty} \beta^t y_t^2 \mid J_0 < +\infty]$$

For **pricing functionals**, we express values as inner products

$$\pi(c) = E \sum_{t=0}^{\infty} \beta^t p_t^0 \cdot c_t \mid J_0$$

where p_t^0 belongs to L_0^2.

With these objects in our toolkit, we move on to state the problem of a **Representative Household in a competitive equilibrium**.

19.1.19 Representative Household

The representative household owns endowment process and initial stocks of h and k and chooses stochastic processes for $\{c_t,\, s_t,\, h_t,\, \ell_t\}_{t=0}^{\infty}$, each element of which is in L_0^2, to maximize

$$-\frac{1}{2}\, E_0 \sum_{t=0}^{\infty} \beta^t \left[(s_t - b_t) \cdot (s_t - b_t) + \ell_t^2 \right]$$

subject to

$$E \sum_{t=0}^{\infty} \beta^t\, p_t^0 \cdot c_t \mid J_0 = E \sum_{t=0}^{\infty} \beta^t \left(w_t^0 \ell_t + \alpha_t^0 \cdot d_t \right) \mid J_0 + v_0 \cdot k_{-1}$$

$$s_t = \Lambda h_{t-1} + \Pi c_t$$

$$h_t = \Delta_h h_{t-1} + \Theta_h c_t, \quad h_{-1}, k_{-1} \text{ given}$$

We now describe the problems faced by two types of firms called type I and type II.

19.1.20 Type I Firm

A type I firm rents capital and labor and endowments and produces c_t, i_t.

It chooses stochastic processes for $\{c_t, i_t, k_t, \ell_t, g_t, d_t\}$, each element of which is in L_0^2, to maximize

$$E_0 \sum_{t=0}^{\infty} \beta^t \left(p_t^0 \cdot c_t + q_t^0 \cdot i_t - r_t^0 \cdot k_{t-1} - w_t^0 \ell_t - \alpha_t^0 \cdot d_t \right)$$

subject to

$$\Phi_c\, c_t + \Phi_g\, g_t + \Phi_i\, i_t = \Gamma k_{t-1} + d_t$$

$$-\ell_t^2 + g_t \cdot g_t = 0$$

19.1.21 Type II Firm

A firm of type II acquires capital via investment and then rents stocks of capital to the c, i-producing type I firm.

A type II firm is a price taker facing the vector v_0 and the stochastic processes $\{r_t^0, q_t^0\}$.

The firm chooses k_{-1} and stochastic processes for $\{k_t, i_t\}_{t=0}^{\infty}$ to maximize

$$E \sum_{t=0}^{\infty} \beta^t (r_t^0 \cdot k_{t-1} - q_t^0 \cdot i_t) \mid J_0 - v_0 \cdot k_{-1}$$

subject to

$$k_t = \Delta_k k_{t-1} + \Theta_k i_t$$

19.1.22 Competitive Equilibrium: Definition

We can now state the following.

Definition: A competitive equilibrium is a price system $[v_0, \{p_t^0, w_t^0, \alpha_t^0, q_t^0, r_t^0\}_{t=0}^\infty]$ and an allocation $\{c_t, i_t, k_t, h_t, g_t, d_t\}_{t=0}^\infty$ that satisfy the following conditions:

- Each component of the price system and the allocation resides in the space L_0^2.
- Given the price system and given h_{-1}, k_{-1}, the allocation solves the representative household's problem and the problems of the two types of firms.

Versions of the two classical welfare theorems prevail under our assumptions.

We exploit that fact in our algorithm for computing a competitive equilibrium.

Step 1: Solve the planning problem by using dynamic programming.

The allocation (i.e., **quantities**) that solve the planning problem **are** the competitive equilibrium quantities.

Step 2: use the following formulas to compute the **equilibrium price system**

$$p_t^0 = [\Pi' M_t^s + \Theta_h' M_t^h]/\mu_0^w = M_t^c/\mu_0^w$$

$$w_t^0 = \mid S_g x_t \mid /\mu_0^w$$

$$r_t^0 = \Gamma' M_t^d/\mu_0^w$$

$$q_t^0 = \Theta_k' M_t^k/\mu_0^w = M_t^i/\mu_0^w$$

$$\alpha_t^0 = M_t^d/\mu_0^w$$

$$v_0 = \Gamma' M_0^d/\mu_0^w + \Delta_k' M_0^k/\mu_0^w$$

Verification: With this price system, values can be assigned to the Lagrange multipliers for each of our three classes of agents that cause all first-order necessary conditions to be satisfied at these prices and at the quantities associated with the optimum of the planning problem.

19.1.23 Asset pricing

An important use of an equilibrium pricing system is to do asset pricing.

Thus, imagine that we are presented a dividend stream: $\{y_t\} \in L_0^2$ and want to compute the value of a perpetual claim to this stream.

To value this asset we simply take **price times quantity** and add to get an asset value: $a_0 = E \sum_{t=0}^\infty \beta^t \, p_t^0 \cdot y_t \mid J_0$.

To compute ao we proceed as follows.

We let

$$y_t = U_a \, x_t$$

$$a_0 = E \sum_{t=0}^\infty \beta^t \, x_t' \, Z_a x_t \mid J_0$$

$$Z_a = U_a' M_c/\mu_0^w$$

We have the following convenient formulas:

$$a_0 = x_0' \, \mu_a \, x_0 + \sigma_a$$

$$\mu_a = \sum_{\tau=0}^{\infty} \beta^{\tau} \left(A^{o\prime}\right)^{\tau} Z_a A^{o\tau}$$

$$\sigma_a = \frac{\beta}{1-\beta} \text{trace} \left(Z_a \sum_{\tau=0}^{\infty} \beta^{\tau} \left(A^{o}\right)^{\tau} CC' (A^{o\prime})^{\tau} \right)$$

19.1.24 Re-Opening Markets

We have assumed that all trading occurs once-and-for-all at time $t = 0$.

If we were to **re-open markets** at some time $t > 0$ at time t wealth levels implicitly defined by time 0 trades, we would obtain the same equilibrium allocation (i.e., quantities) and the following time t price system

$$L_t^2 = [\{y_s\}_{s=t}^{\infty} : \ y_s \ \text{ is a random variable in } \ J_s \ \text{ for } \ s \geq t$$
$$\text{and } E \sum_{s=t}^{\infty} \beta^{s-t} \, y_s^2 \mid J_t < +\infty].$$

$$p_s^t = M_c x_s / [\bar{e}_j M_c x_t], \qquad s \geq t$$
$$w_s^t = \mid S_g x_s | / [\bar{e}_j M_c x_t], \quad s \geq t$$
$$r_s^t = \Gamma' M_d x_s / [\bar{e}_j M_c x_t], \quad s \geq t$$
$$q_s^t = M_i x_s / [\bar{e}_j \, M_c x_t], \qquad s \geq t$$
$$\alpha_s^t = M_d x_s / [\bar{e}_j \, M_c x_t], \quad s \geq t$$
$$v_t = [\Gamma' M_d + \Delta_k' M_k] x_t / \, [\bar{e}_j \, M_c x_t]$$

19.2 Econometrics

Up to now, we have described how to solve the **direct problem** that maps model parameters into an (equilibrium) stochastic process of prices and quantities.

Recall the **inverse problem** of inferring model parameters from a single realization of a time series of some of the prices and quantities.

Another name for the inverse problem is **econometrics**.

An advantage of the [Hansen and Sargent, 2013] structure is that it comes with a self-contained theory of econometrics.

It is really just a tale of two state-space representations.

Here they are:

Original State-Space Representation:

$$x_{t+1} = A^o x_t + C w_{t+1}$$
$$y_t = G x_t + v_t$$

where v_t is a martingale difference sequence of measurement errors that satisfies $E v_t v_t' = R, E w_{t+1} v_s' = 0$ for all $t + 1 \geq s$ and

$$x_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_0)$$

Innovations Representation:

$$\begin{aligned}\hat{x}_{t+1} &= A^o \hat{x}_t + K_t a_t \\ y_t &= G\hat{x}_t + a_t,\end{aligned}$$

where $a_t = y_t - E[y_t|y^{t-1}], Ea_t a_t' \equiv \Omega_t = G\Sigma_t G' + R.$

Compare numbers of shocks in the two representations:

- $n_w + n_y$ versus n_y

Compare spaces spanned

- $H(y^t) \subset H(w^t, v^t)$
- $H(y^t) = H(a^t)$

Kalman Filter:.

Kalman gain:

$$K_t = A^o \Sigma_t G' (G\Sigma_t G' + R)^{-1}$$

Riccati Difference Equation:

$$\begin{aligned}\Sigma_{t+1} &= A^o \Sigma_t A^{o\prime} + CC' \\ &- A^o \Sigma_t G' (G\Sigma_t G' + R)^{-1} G\Sigma_t A^{o\prime}\end{aligned}$$

Innovations Representation as Whitener

Whitening Filter:

$$\begin{aligned}a_t &= y_t - G\hat{x}_t \\ \hat{x}_{t+1} &= A^o \hat{x}_t + K_t a_t\end{aligned}$$

can be used recursively to construct a record of innovations $\{a_t\}_{t=0}^T$ from an $(\hat{x}_0, \Sigma_0)$ and a record of observations $\{y_t\}_{t=0}^T$.

Limiting Time-Invariant Innovations Representation

$$\begin{aligned}\Sigma &= A^o \Sigma A^{o\prime} + CC' \\ &- A^o \Sigma G' (G\Sigma G' + R)^{-1} G\Sigma A^{o\prime} \\ K &= A^o \Sigma_t G' (G\Sigma G' + R)^{-1}\end{aligned}$$

$$\begin{aligned}\hat{x}_{t+1} &= A^o \hat{x}_t + K a_t \\ y_t &= G\hat{x}_t + a_t\end{aligned}$$

where $Ea_t a_t' \equiv \Omega = G\Sigma G' + R.$

19.2.1 Factorization of Likelihood Function

Sample of observations $\{y_s\}_{s=0}^T$ on a $(n_y \times 1)$ vector.

$$\begin{aligned}f(y_T, y_{T-1}, \ldots, y_0) &= f_T(y_T|y_{T-1}, \ldots, y_0) f_{T-1}(y_{T-1}|y_{T-2}, \ldots, y_0) \cdots f_1(y_1|y_0) f_0(y_0) \\ &= g_T(a_T) g_{T-1}(a_{T-1}) \ldots g_1(a_1) f_0(y_0).\end{aligned}$$

Gaussian Log-Likelihood:

$$-.5 \sum_{t=0}^T \left\{ n_y \ln(2\pi) + \ln |\Omega_t| + a_t' \Omega_t^{-1} a_t \right\}$$

19.2.2 Covariance Generating Functions

Autocovariance: $C_x(\tau) = Ex_t x'_{t-\tau}$.

Generating Function: $S_x(z) = \sum_{\tau=-\infty}^{\infty} C_x(\tau) z^\tau, z \in C$.

19.2.3 Spectral Factorization Identity

Original state-space representation has too many shocks and implies:

$$S_y(z) = G(zI - A^o)^{-1} CC'(z^{-1}I - (A^o)')^{-1} G' + R$$

Innovations representation has as many shocks as dimension of y_t and implies

$$S_y(z) = [G(zI - A^o)^{-1} K + I][G\Sigma G' + R][K'(z^{-1}I - A^{o\prime})^{-1} G' + I]$$

Equating these two leads to:

$$\begin{aligned}&G(zI - A^o)^{-1} CC'(z^{-1}I - A^{o\prime})^{-1} G' + R = \\ &[G(zI - A^o)^{-1} K + I][G\Sigma G' + R][K'(z^{-1}I - A^{o\prime})^{-1} G' + I].\end{aligned}$$

Key Insight: The zeros of the polynomial $\det[G(zI - A^o)^{-1} K + I]$ all lie inside the unit circle, which means that a_t lies in the space spanned by square summable linear combinations of y^t.

$$H(a^t) = H(y^t)$$

Key Property: Invertibility

19.2.4 Wold and Vector Autoregressive Representations

Let's start with some lag operator arithmetic.

The lag operator L and the inverse lag operator L^{-1} each map an infinite sequence into an infinite sequence according to the transformation rules

$$Lx_t \equiv x_{t-1}$$

$$L^{-1}x_t \equiv x_{t+1}$$

A **Wold moving average representation** for $\{y_t\}$ is

$$y_t = [G(I - A^o L)^{-1} KL + I] a_t$$

Applying the inverse of the operator on the right side and using

$$[G(I - A^o L)^{-1} KL + I]^{-1} = I - G[I - (A^o - KG)L]^{-1} KL$$

gives the **vector autoregressive representation**

$$y_t = \sum_{j=1}^{\infty} G(A^o - KG)^{j-1} K y_{t-j} + a_t$$

19.3 Dynamic Demand Curves and Canonical Household Technologies

19.3.1 Canonical Household Technologies

$$
\begin{aligned}
h_t &= \Delta_h h_{t-1} + \Theta_h c_t \\
s_t &= \Lambda h_{t-1} + \Pi c_t \\
b_t &= U_b z_t
\end{aligned}
$$

Definition: A household service technology $(\Delta_h, \Theta_h, \Pi, \Lambda, U_b)$ is said to be **canonical** if

- Π is nonsingular, and
- the absolute values of the eigenvalues of $(\Delta_h - \Theta_h \Pi^{-1} \Lambda)$ are strictly less than $1/\sqrt{\beta}$.

Key invertibility property: A canonical household service technology maps a service process $\{s_t\}$ in L_0^2 into a corresponding consumption process $\{c_t\}$ for which the implied household capital stock process $\{h_t\}$ is also in L_0^2.

An inverse household technology:

$$
\begin{aligned}
c_t &= -\Pi^{-1} \Lambda h_{t-1} + \Pi^{-1} s_t \\
h_t &= (\Delta_h - \Theta_h \Pi^{-1} \Lambda) h_{t-1} + \Theta_h \Pi^{-1} s_t
\end{aligned}
$$

The restriction on the eigenvalues of the matrix $(\Delta_h - \Theta_h \Pi^{-1} \Lambda)$ keeps the household capital stock $\{h_t\}$ in L_0^2.

19.3.2 Dynamic Demand Functions

$$
\rho_t^0 \equiv \Pi^{-1\prime} \Big[p_t^0 - \Theta_h' E_t \sum_{\tau=1}^\infty \beta^\tau (\Delta_h' - \Lambda' \Pi^{-1\prime} \Theta_h')^{\tau-1} \Lambda' \Pi^{-1\prime} p_{t+\tau}^0 \Big]
$$

$$
\begin{aligned}
s_{i,t} &= \Lambda h_{i,t-1} \\
h_{i,t} &= \Delta_h h_{i,t-1}
\end{aligned}
$$

where $h_{i,-1} = h_{-1}$.

$$
W_0 = E_0 \sum_{t=0}^\infty \beta^t (w_t^0 \ell_t + \alpha_t^0 \cdot d_t) + v_0 \cdot k_{-1}
$$

$$
\mu_0^w = \frac{E_0 \sum_{t=0}^\infty \beta^t \rho_t^0 \cdot (b_t - s_{i,t}) - W_0}{E_0 \sum_{t=0}^\infty \beta^t \rho_t^0 \cdot \rho_t^0}
$$

$$
\begin{aligned}
c_t &= -\Pi^{-1} \Lambda h_{t-1} + \Pi^{-1} b_t - \Pi^{-1} \mu_0^w E_t \{ \Pi'^{-1} - \Pi'^{-1} \Theta_h' \\
&\qquad [I - (\Delta_h' - \Lambda' \Pi'^{-1} \Theta_h') \beta L^{-1}]^{-1} \Lambda' \Pi'^{-1} \beta L^{-1} \} p_t^0 \\
h_t &= \Delta_h h_{t-1} + \Theta_h c_t
\end{aligned}
$$

This system expresses consumption demands at date t as functions of: (i) time-t conditional expectations of future scaled Arrow-Debreu prices $\{p_{t+s}^0\}_{s=0}^\infty$; (ii) the stochastic process for the household's endowment $\{d_t\}$ and preference shock $\{b_t\}$, as mediated through the multiplier μ_0^w and wealth W_0; and (iii) past values of consumption, as mediated through the state variable h_{t-1}.

19.4 Gorman Aggregation and Engel Curves

We shall explore how the dynamic demand schedule for consumption goods opens up the possibility of satisfying Gorman's (1953) conditions for aggregation in a heterogeneous consumer model.

The first equation of our demand system is an Engel curve for consumption that is linear in the marginal utility μ_0^2 of individual wealth with a coefficient on μ_0^w that depends only on prices.

The multiplier μ_0^w depends on wealth in an affine relationship, so that consumption is linear in wealth.

In a model with multiple consumers who have the same household technologies $(\Delta_h, \Theta_h, \Lambda, \Pi)$ but possibly different preference shock processes and initial values of household capital stocks, the coefficient on the marginal utility of wealth is the same for all consumers.

Gorman showed that when Engel curves satisfy this property, there exists a unique community or aggregate preference ordering over aggregate consumption that is independent of the distribution of wealth.

19.4.1 Re-Opened Markets

$$\rho_t^t \equiv \Pi^{-1\prime}\Big[p_t^t - \Theta_h' E_t \sum_{\tau=1}^{\infty} \beta^\tau (\Delta_h' - \Lambda'\Pi^{-1\prime}\Theta_h')^{\tau-1}\Lambda'\Pi^{-1\prime}p_{t+\tau}^t\Big]$$

$$s_{i,t} = \Lambda h_{i,t-1}$$

$$h_{i,t} = \Delta_h h_{i,t-1},$$

where now $h_{i,t-1} = h_{t-1}$. Define time t wealth W_t

$$W_t = E_t \sum_{j=0}^{\infty} \beta^j (w_{t+j}^t \ell_{t+j} + \alpha_{t+j}^t \cdot d_{t+j}) + v_t \cdot k_{t-1}$$

$$\mu_t^w = \frac{E_t \sum_{j=0}^{\infty} \beta^j \rho_{t+j}^t \cdot (b_{t+j} - s_{i,t+j}) - W_t}{E_t \sum_{t=0}^{\infty} \beta^j \rho_{t+j}^t \cdot \rho_{t+j}^t}$$

$$\begin{aligned} c_t &= -\Pi^{-1}\Lambda h_{t-1} + \Pi^{-1}b_t - \Pi^{-1}\mu_t^w E_t\{\Pi'^{-1} - \Pi'^{-1}\Theta_h' \\ &\quad [I - (\Delta_h' - \Lambda'\Pi'^{-1}\Theta_h')\beta L^{-1}]^{-1}\Lambda'\Pi'^{-1}\beta L^{-1}\}p_t^t \\ h_t &= \Delta_h h_{t-1} + \Theta_h c_t \end{aligned}$$

19.4.2 Dynamic Demand

Define a time t continuation of a sequence $\{z_t\}_{t=0}^{\infty}$ as the sequence $\{z_\tau\}_{\tau=t}^{\infty}$. The demand system indicates that the time t vector of demands for c_t is influenced by:

Through the multiplier μ_t^w, the time t continuation of the preference shock process $\{b_t\}$ and the time t continuation of $\{s_{i,t}\}$.

The time $t-1$ level of household durables h_{t-1}.

Everything that affects the household's time t wealth, including its stock of physical capital k_{t-1} and its value v_t, the time t continuation of the factor prices $\{w_t, \alpha_t\}$, the household's continuation endowment process, and the household's continuation plan for $\{\ell_t\}$.

The time t continuation of the vector of prices $\{p_t^t\}$.

19.4.3 Attaining a Canonical Household Technology

Apply the following version of a factorization identity:

$$\begin{aligned}[\Pi + \beta^{1/2} L^{-1} \Lambda (I - \beta^{1/2} L^{-1} \Delta_h)^{-1} \Theta_h]' [\Pi + \beta^{1/2} L \Lambda (I - \beta^{1/2} L \Delta_h)^{-1} \Theta_h] \\ = [\hat\Pi + \beta^{1/2} L^{-1} \hat\Lambda (I - \beta^{1/2} L^{-1} \Delta_h)^{-1} \Theta_h]' [\hat\Pi + \beta^{1/2} L \hat\Lambda (I - \beta^{1/2} L \Delta_h)^{-1} \Theta_h]\end{aligned}$$

The factorization identity guarantees that the $[\hat\Lambda, \hat\Pi]$ representation satisfies both requirements for a canonical representation.

19.5 Partial Equilibrium

Now we'll provide quick overviews of examples of economies that fit within our framework

We provide details for a number of these examples in subsequent lectures

1. *Growth in Dynamic Linear Economies*
2. *Lucas Asset Pricing using DLE*
3. *IRFs in Hall Model*
4. *Permanent Income Using the DLE class*
5. *Rosen schooling model*
6. *Cattle cycles*
7. *Shock Non Invertibility*

We'll start with an example of a **partial equilibrium** in which we posit demand and supply curves

Suppose that we want to capture the dynamic demand curve:

$$\begin{aligned}c_t &= -\Pi^{-1} \Lambda h_{t-1} + \Pi^{-1} b_t - \Pi^{-1} \mu_0^w E_t \{ \Pi'^{-1} - \Pi'^{-1} \Theta_h' \\ &\quad [I - (\Delta_h' - \Lambda' \Pi'^{-1} \Theta_h') \beta L^{-1}]^{-1} \Lambda' \Pi'^{-1} \beta L^{-1} \} p_t \\ h_t &= \Delta_h h_{t-1} + \Theta_h c_t\end{aligned}$$

From material described earlier in this lecture, we know how to reverse engineer preferences that generate this demand system

- note how the demand equations are cast in terms of the matrices in our standard preference representation

Now let's turn to supply.

A representative firm takes as given and beyond its control the stochastic process $\{p_t\}_{t=0}^\infty$.

The firm sells its output c_t in a competitive market each period.

Only spot markets convene at each date $t \geq 0$.

The firm also faces an exogenous process of cost disturbances d_t.

The firm chooses stochastic processes $\{c_t, g_t, i_t, k_t\}_{t=0}^\infty$ to maximize

$$E_0 \sum_{t=0}^\infty \beta^t \{ p_t \cdot c_t - g_t \cdot g_t / 2 \}$$

subject to given k_{-1} and

$$\begin{aligned}\Phi_c c_t + \Phi_i i_t + \Phi_g g_t &= \Gamma k_{t-1} + d_t \\ k_t &= \Delta_k k_{t-1} + \Theta_k i_t.\end{aligned}$$

19.6 Equilibrium Investment Under Uncertainty

A representative firm maximizes

$$E\sum_{t=0}^{\infty}\beta^t\{p_tc_t-g_t^2/2\}$$

subject to the technology

$$\begin{aligned}c_t &= \gamma k_{t-1}\\ k_t &= \delta_k k_{t-1}+i_t\\ g_t &= f_1 i_t + f_2 d_t\end{aligned}$$

where d_t is a cost shifter, $\gamma > 0$, and $f_1 > 0$ is a cost parameter and $f_2 = 1$. Demand is governed by

$$p_t = \alpha_0 - \alpha_1 c_t + u_t$$

where u_t is a demand shifter with mean zero and α_0, α_1 are positive parameters.

Assume that u_t, d_t are uncorrelated first-order autoregressive processes.

19.7 A Rosen-Topel Housing Model

$$\begin{aligned}R_t &= b_t + \alpha h_t\\ p_t &= E_t\sum_{\tau=0}^{\infty}(\beta\delta_h)^{\tau}R_{t+\tau}\end{aligned}$$

where h_t is the stock of housing at time t R_t is the rental rate for housing, p_t is the price of new houses, and b_t is a demand shifter; $\alpha < 0$ is a demand parameter, and δ_h is a depreciation factor for houses.

We cast this demand specification within our class of models by letting the stock of houses h_t evolve according to

$$h_t = \delta_h h_{t-1} + c_t, \quad \delta_h \in (0,1)$$

where c_t is the rate of production of new houses.

Houses produce services s_t according to $s_t = \bar{\lambda} h_t$ or $s_t = \lambda h_{t-1} + \pi c_t$, where $\lambda = \bar{\lambda}\delta_h, \pi = \bar{\lambda}$.

We can take $\bar{\lambda}\rho_t^0 = R_t$ as the rental rate on housing at time t, measured in units of time t consumption (housing).

Demand for housing services is

$$s_t = b_t - \mu_0\rho_t^0$$

where the price of new houses p_t is related to ρ_t^0 by $\rho_t^0 = \pi^{-1}[p_t - \beta\delta_h E_t p_{t+1}]$.

19.8 Cattle Cycles

Rosen, Murphy, and Scheinkman (1994). Let p_t be the price of freshly slaughtered beef, m_t the feeding cost of preparing an animal for slaughter, $\tilde{h}_t$ the one-period holding cost for a mature animal, $\gamma_1\tilde{h}_t$ the one-period holding cost for a yearling, and $\gamma_0\tilde{h}_t$ the one-period holding cost for a calf.

The cost processes $\{\tilde{h}_t, m_t\}_{t=0}^{\infty}$ are exogenous, while the stochastic process $\{p_t\}_{t=0}^{\infty}$ is determined by a rational expectations equilibrium. Let $\tilde{x}_t$ be the breeding stock, and $\tilde{y}_t$ be the total stock of animals.

The law of motion for cattle stocks is

$$\tilde{x}_t = (1-\delta)\tilde{x}_{t-1} + g\tilde{x}_{t-3} - c_t$$

where c_t is a rate of slaughtering. The total head-count of cattle

$$\tilde{y}_t = \tilde{x}_t + g\tilde{x}_{t-1} + g\tilde{x}_{t-2}$$

is the sum of adults, calves, and yearlings, respectively.

A representative farmer chooses $\{c_t, \tilde{x}_t\}$ to maximize

$$E_0 \sum_{t=0}^{\infty} \beta^t \{p_t c_t - \tilde{h}_t \tilde{x}_t - (\gamma_0 \tilde{h}_t)(g\tilde{x}_{t-1}) - (\gamma_1 \tilde{h}_t)(g\tilde{x}_{t-2}) - m_t c_t \\ - \Psi(\tilde{x}_t, \tilde{x}_{t-1}, \tilde{x}_{t-2}, c_t)\}$$

where

$$\Psi = \frac{\psi_1}{2}\tilde{x}_t^2 + \frac{\psi_2}{2}\tilde{x}_{t-1}^2 + \frac{\psi_3}{2}\tilde{x}_{t-2}^2 + \frac{\psi_4}{2}c_t^2$$

Demand is governed by

$$c_t = \alpha_0 - \alpha_1 p_t + \tilde{d}_t$$

where $\alpha_0 > 0$, $\alpha_1 > 0$, and $\{\tilde{d}_t\}_{t=0}^{\infty}$ is a stochastic process with mean zero representing a demand shifter.

For more details see *Cattle cycles*

19.9 Models of Occupational Choice and Pay

We'll describe the following pair of schooling models that view education as a time-to-build process:

- Rosen schooling model for engineers
- Two-occupation model

19.9.1 Market for Engineers

Ryoo and Rosen's (2004) [Ryoo and Rosen, 2004] model consists of the following equations:

first, a demand curve for engineers

$$w_t = -\alpha_d N_t + \epsilon_{1t} \ , \ \alpha_d > 0$$

second, a time-to-build structure of the education process

$$N_{t+k} = \delta_N N_{t+k-1} + n_t \ , \ 0 < \delta_N < 1$$

third, a definition of the discounted present value of each new engineering student

$$v_t = \beta^k E_t \sum_{j=0}^{\infty} (\beta\delta_N)^j w_{t+k+j};$$

and fourth, a supply curve of new students driven by v_t

$$n_t = \alpha_s v_t + \epsilon_{2t} \ , \ \alpha_s > 0$$

Here $\{\epsilon_{1t}, \epsilon_{2t}\}$ are stochastic processes of labor demand and supply shocks.

Definition: A partial equilibrium is a stochastic process $\{w_t, N_t, v_t, n_t\}_{t=0}^{\infty}$ satisfying these four equations, and initial conditions $N_{-1}, n_{-s}, s = 1, \ldots, -k$.

We sweep the time-to-build structure and the demand for engineers into the household technology and putting the supply of new engineers into the technology for producing goods.

$$s_t = [\lambda_1 \; 0 \; \ldots \; 0] \begin{bmatrix} h_{1t-1} \\ h_{2t-1} \\ \vdots \\ h_{k+1,t-1} \end{bmatrix} + 0 \cdot c_t$$

$$\begin{bmatrix} h_{1t} \\ h_{2t} \\ \vdots \\ h_{k,t} \\ h_{k+1,t} \end{bmatrix} = \begin{bmatrix} \delta_N & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} h_{1t-1} \\ h_{2t-1} \\ \vdots \\ h_{k,t-1} \\ h_{k+1,t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} c_t$$

This specification sets Rosen's $N_t = h_{1t-1}, n_t = c_t, h_{\tau+1,t-1} = n_{t-\tau}, \tau = 1, \ldots, k$, and uses the home-produced service to capture the demand for labor. Here λ_1 embodies Rosen's demand parameter α_d.

- The supply of new workers becomes our consumption.
- The dynamic demand curve becomes Rosen's dynamic supply curve for new workers.

Remark: This has an Imai-Keane flavor.

For more details and Python code see *Rosen schooling model*.

19.9.2 Skilled and Unskilled Workers

First, a demand curve for labor

$$\begin{bmatrix} w_{ut} \\ w_{st} \end{bmatrix} = \alpha_d \begin{bmatrix} N_{ut} \\ N_{st} \end{bmatrix} + \epsilon_{1t}$$

where α_d is a (2×2) matrix of demand parameters and ϵ_{1t} is a vector of demand shifters second, time-to-train specifications for skilled and unskilled labor, respectively:

$$N_{st+k} = \delta_N N_{st+k-1} + n_{st}$$
$$N_{ut} = \delta_N N_{ut-1} + n_{ut};$$

where N_{st}, N_{ut} are stocks of the two types of labor, and n_{st}, n_{ut} are entry rates into the two occupations.

third, definitions of discounted present values of new entrants to the skilled and unskilled occupations, respectively:

$$v_{st} = E_t \beta^k \sum_{j=0}^{\infty} (\beta \delta_N)^j w_{st+k+j}$$

$$v_{ut} = E_t \sum_{j=0}^{\infty} (\beta \delta_N)^j w_{ut+j}$$

where w_{ut}, w_{st} are wage rates for the two occupations; and fourth, supply curves for new entrants:

$$\begin{bmatrix} n_{st} \\ n_{ut} \end{bmatrix} = \alpha_s \begin{bmatrix} v_{ut} \\ v_{st} \end{bmatrix} + \epsilon_{2t}$$

Short Cut

As an alternative, Siow simply used the **equalizing differences** condition

$$v_{ut} = v_{st}$$

19.10 Permanent Income Models

We'll describe a class of permanent income models that feature

- Many consumption goods and services
- A single capital good with $R\beta = 1$
- The physical production technology

$$\begin{aligned} \phi_c \cdot c_t + i_t &= \gamma k_{t-1} + e_t \\ k_t &= k_{t-1} + i_t \end{aligned}$$

$$\phi_i i_t - g_t = 0$$

Implication One:

Equality of Present Values of Moving Average Coefficients of c and e

$$k_{t-1} = \beta \sum_{j=0}^{\infty} \beta^j (\phi_c \cdot c_{t+j} - e_{t+j}) \quad \Rightarrow$$

$$k_{t-1} = \beta \sum_{j=0}^{\infty} \beta^j E(\phi_c \cdot c_{t+j} - e_{t+j}) | J_t \quad \Rightarrow$$

$$\sum_{j=0}^{\infty} \beta^j (\phi_c)' \chi_j = \sum_{j=0}^{\infty} \beta^j \epsilon_j$$

where $\chi_j w_t$ is the response of c_{t+j} to w_t and $\epsilon_j w_t$ is the response of endowment e_{t+j} to w_t:

Implication Two:

Martingales

$$\begin{aligned} \mathcal{M}_t^k &= E(\mathcal{M}_{t+1}^k | J_t) \\ \mathcal{M}_t^e &= E(\mathcal{M}_{t+1}^e | J_t) \end{aligned}$$

and

$$\mathcal{M}_t^c = (\Phi_c)' \mathcal{M}_t^d = \phi_c M_t^e$$

For more details see *Permanent Income Using the DLE class*

Testing Permanent Income Models:

We have two types of implications of permanent income models:

- Equality of present values of moving average coefficients.
- Martingale $\mathcal{M}_t^k$.

These have been tested in work by Hansen, Sargent, and Roberts (1991) [Sargent *et al.*, 1991] and by Attanasio and Pavoni (2011) [Attanasio and Pavoni, 2011].

19.11 Gorman Heterogeneous Households

We now assume that there is a finite number of households, each with its own household technology and preferences over consumption services.

Household j orders preferences over consumption processes according to

$$-\left(\frac{1}{2}\right) E \sum_{t=0}^{\infty} \beta^t \left[(s_{jt} - b_{jt}) \cdot (s_{jt} - b_{jt}) + \ell_{jt}^2\right] \mid J_0$$

$$s_{jt} = \Lambda\, h_{j,t-1} + \Pi\, c_{jt}$$

$$h_{jt} = \Delta_h\, h_{j,t-1} + \Theta_h\, c_{jt}$$

and $h_{j,-1}$ is given

$$b_{jt} = U_{bj} z_t$$

$$E \sum_{t=0}^{\infty} \beta^t\, p_t^0 \cdot c_{jt} \mid J_0 = E \sum_{t=0}^{\infty} \beta^t\, (w_t^0\, \ell_{jt} + \alpha_t^0 \cdot d_{jt}) \mid J_0 + v_0 \cdot k_{j,-1},$$

where $k_{j,-1}$ is given. The j^{th} consumer owns an endowment process d_{jt}, governed by the stochastic process $d_{jt} = U_{dj}\, z_t$.

We refer to this as a setting with Gorman heterogeneous households.

This specification confines heterogeneity among consumers to:

- differences in the preference processes $\{b_{jt}\}$, represented by different selections of U_{bj}
- differences in the endowment processes $\{d_{jt}\}$, represented by different selections of U_{dj}
- differences in $h_{j,-1}$ and
- differences in $k_{j,-1}$

The matrices Λ, Π, Δ_h, Θ_h do not depend on j.

This makes everybody's demand system have the form described earlier, with different μ_{j0}^w's (reflecting different wealth levels) and different b_{jt} preference shock processes and initial conditions for household capital stocks.

Punchline: there exists a representative consumer.

We can use the representative consumer to compute a competitive equilibrium **aggregate** allocation and price system.

With the equilibrium aggregate allocation and price system in hand, we can then compute allocations to each household.

Computing Allocations to Individuals:

Set

$$\ell_{jt} = (\mu_{0j}^w / \mu_{0a}^w) \ell_{at}$$

Then solve the following equation for μ_{0j}^w:

$$\mu_{0j}^w E_0 \sum_{t=0}^{\infty} \beta^t \{\rho_t^0 \cdot \rho_t^0 + (w_t^0 / \mu_{0a}^w) \ell_{at}\} = E_0 \sum_{t=0}^{\infty} \beta^t \{\rho_t^0 \cdot (b_{jt} - s_{jt}^i) - \alpha_t^0 \cdot d_{jt}\} - v_0 k_{j,-1}$$

$$s_{jt} - b_{jt} = \mu_{0j}^w \rho_t^0$$

$$\begin{aligned} c_{jt} &= -\Pi^{-1} \Lambda h_{j,t-1} + \Pi^{-1} s_{jt} \\ h_{jt} &= (\Delta_h - \Theta_h \Pi^{-1} \Lambda) h_{j,t-1} + \Pi^{-1} \Theta_h s_{jt} \end{aligned}$$

Here $h_{j,-1}$ given.

19.12 Non-Gorman Heterogeneous Households

We now describe a less tractable type of heterogeneity across households that we dub **Non-Gorman heterogeneity**.

Here is the specification:

Preferences and Household Technologies:

$$-\frac{1}{2}E\sum_{t=0}^{\infty}\beta^t\left[(s_{it}-b_{it})\cdot(s_{it}-b_{it})+\ell_{it}^2\right]\mid J_0$$

$$\begin{aligned} s_{it} &= \Lambda_i h_{it-1}+\Pi_i\, c_{it}\\ h_{it} &= \Delta_{h_i}\, h_{it-1}+\Theta_{h_i} c_{it}\ ,\ i=1,2. \end{aligned}$$

$$b_{it}=U_{bi}z_t$$

$$z_{t+1}=A_{22}z_t+C_2 w_{t+1}$$

Production Technology

$$\Phi_c(c_{1t}+c_{2t})+\Phi_g g_t+\Phi_i i_t=\Gamma k_{t-1}+d_{1t}+d_{2t}$$

$$k_t=\Delta_k k_{t-1}+\Theta_k i_t$$

$$g_t\cdot g_t=\ell_t^2,\qquad \ell_t=\ell_{1t}+\ell_{2t}$$

$$d_{it}=U_{d_i}z_t,\quad i=1,2$$

Pareto Problem:

$$\begin{aligned} &-\frac{1}{2}\lambda E_0\sum_{t=0}^{\infty}\beta^t[(s_{1t}-b_{1t})\cdot(s_{1t}-b_{1t})+\ell_{1t}^2]\\ &-\frac{1}{2}(1-\lambda)E_0\sum_{t=0}^{\infty}\beta^t[(s_{2t}-b_{2t})\cdot(s_{2t}-b_{2t})+\ell_{2t}^2] \end{aligned}$$

Mongrel Aggregation: Static

There is what we call a kind of **mongrel aggregation** in this setting.

We first describe the idea within a simple static setting in which there is a single consumer static inverse demand with implied preferences:

$$c_t=\Pi^{-1}b_t-\mu_0\Pi^{-1}\Pi^{-1\prime}p_t$$

An inverse demand curve is

$$p_t=\mu_0^{-1}\Pi' b_t-\mu_0^{-1}\Pi'\Pi c_t$$

Integrating the marginal utility vector shows that preferences can be taken to be

$$(-2\mu_0)^{-1}(\Pi c_t-b_t)\cdot(\Pi c_t-b_t)$$

Key Insight: Factor the inverse of a 'covariance matrix'.

Now assume that there are two consumers, $i=1,2$, with demand curves

$$c_{it}=\Pi_i^{-1}b_{it}-\mu_{0i}\Pi_i^{-1}\Pi_i^{-1\prime}p_t$$

$$c_{1t} + c_{2t} = (\Pi_1^{-1} b_{1t} + \Pi_2^{-1} b_{2t}) - (\mu_{01} \Pi_1^{-1} \Pi_1^{-1\prime} + \mu_{02} \Pi_2 \Pi_2^{-1\prime}) p_t$$

Setting $c_{1t} + c_{2t} = c_t$ and solving for p_t gives

$$\begin{aligned} p_t &= (\mu_{01} \Pi_1^{-1} \Pi_1^{-1\prime} + \mu_{02} \Pi_2^{-1} \Pi_2^{-1\prime})^{-1} (\Pi_1^{-1} b_{1t} + \Pi_2^{-1} b_{2t}) \\ &\quad - (\mu_{01} \Pi_1^{-1} \Pi_1^{-1\prime} + \mu_{02} \Pi_2^{-1} \Pi_2^{-1\prime})^{-1} c_t \end{aligned}$$

Punchline: choose Π associated with the aggregate ordering to satisfy

$$\mu_0^{-1} \Pi' \Pi = (\mu_{01} \Pi_1^{-1} \Pi_2^{-1\prime} + \mu_{02} \Pi_2^{-1} \Pi_2^{-1\prime})^{-1}$$

Dynamic Analogue:

We now describe how to extend mongrel aggregation to a dynamic setting.

The key comparison is

- Static: factor a covariance matrix-like object
- Dynamic: factor a spectral-density matrix-like object

Programming Problem for Dynamic Mongrel Aggregation:

Our strategy for deducing the mongrel preference ordering over $c_t = c_{1t} + c_{2t}$ is to solve the programming problem: choose $\{c_{1t}, c_{2t}\}$ to maximize the criterion

$$\sum_{t=0}^{\infty} \beta^t [\lambda (s_{1t} - b_{1t}) \cdot (s_{1t} - b_{1t}) + (1 - \lambda)(s_{2t} - b_{2t}) \cdot (s_{2t} - b_{2t})]$$

subject to

$$\begin{aligned} h_{jt} &= \Delta_{hj} \, h_{jt-1} + \Theta_{hj} \, c_{jt}, j = 1, 2 \\ s_{jt} &= \Delta_j h_{jt-1} + \Pi_j c_{jt} \;, j = 1, 2 \\ c_{1t} + c_{2t} &= c_t \end{aligned}$$

subject to $(h_{1,-1}, \, h_{2,-1})$ given and $\{b_{1t}\}$, $\{b_{2t}\}$, $\{c_t\}$ being known and fixed sequences.

Substituting the $\{c_{1t}, \, c_{2t}\}$ sequences that solve this problem as functions of $\{b_{1t}, \, b_{2t}, \, c_t\}$ into the objective determines a mongrel preference ordering over $\{c_t\} = \{c_{1t} + c_{2t}\}$.

In solving this problem, it is convenient to proceed by using Fourier transforms. For details, please see [Hansen and Sargent, 2013] where they deploy a

Secret Weapon: Another application of the spectral factorization identity.

Concluding remark: The [Hansen and Sargent, 2013] class of models described in this lecture are all complete markets models. We have exploited the fact that complete market models **are all alike** to allow us to define a class that **gives the same name to different things** in the spirit of Henri Poincare.

Could we create such a class for **incomplete markets** models?

That would be nice, but before trying it would be wise to contemplate the remainder of a statement by Robert E. Lucas, Jr., with which we began this lecture.

> "Complete market economies are all alike but each incomplete market economy is incomplete in its own individual way." Robert E. Lucas, Jr., (1989)

CHAPTER

TWENTY

GROWTH IN DYNAMIC LINEAR ECONOMIES

Contents

This is another member of a suite of lectures that use the quantecon DLE class to instantiate models within the [Hansen and Sargent, 2013] class of models described in detail in *Recursive Models of Dynamic Linear Economies*.

In addition to what's included in Anaconda, this lecture uses the quantecon library.

```
!pip install --upgrade quantecon
```

This lecture describes several complete market economies having a common linear-quadratic-Gaussian structure.

Three examples of such economies show how the DLE class can be used to compute equilibria of such economies in Python and to illustrate how different versions of these economies can or cannot generate sustained growth.

We require the following imports

```
import numpy as np
import matplotlib.pyplot as plt
from quantecon import DLE
```

20.1 Common Structure

Our example economies have the following features

- Information flows are governed by an exogenous stochastic process z_t that follows

$$z_{t+1} = A_{22} z_t + C_2 w_{t+1}$$

  where w_{t+1} is a martingale difference sequence.

- Preference shocks b_t and technology shocks d_t are linear functions of z_t

$$b_t = U_b z_t$$

$$d_t = U_d z_t$$

- Consumption and physical investment goods are produced using the following technology

$$\Phi_c c_t + \Phi_g g_t + \Phi_i i_t = \Gamma k_{t-1} + d_t$$

$$k_t = \Delta_k k_{t-1} + \Theta_k i_t$$

$$g_t \cdot g_t = l_t^2$$

  where c_t is a vector of consumption goods, g_t is a vector of intermediate goods, i_t is a vector of investment goods, k_t is a vector of physical capital goods, and l_t is the amount of labor supplied by the representative household.

- Preferences of a representative household are described by

$$-\frac{1}{2}\mathbb{E}\sum_{t=0}^{\infty}\beta^t[(s_t - b_t)\cdot(s_t - b_t) + l_t^2], 0 < \beta < 1$$

$$s_t = \Lambda h_{t-1} + \Pi c_t$$

$$h_t = \Delta_h h_{t-1} + \Theta_h c_t$$

where s_t is a vector of consumption services, and h_t is a vector of household capital stocks.

Thus, an instance of this class of economies is described by the matrices

$$\{A_{22}, C_2, U_b, U_d, \Phi_c, \Phi_g, \Phi_i, \Gamma, \Delta_k, \Theta_k, \Lambda, \Pi, \Delta_h, \Theta_h\}$$

and the scalar β.

20.2 A Planning Problem

The first welfare theorem asserts that a competitive equilibrium allocation solves the following planning problem.

Choose $\{c_t, s_t, i_t, h_t, k_t, g_t\}_{t=0}^{\infty}$ to maximize

$$-\frac{1}{2}\mathbb{E}\sum_{t=0}^{\infty}\beta^t[(s_t - b_t)\cdot(s_t - b_t) + g_t \cdot g_t]$$

subject to the linear constraints

$$\Phi_c c_t + \Phi_g g_t + \Phi_i i_t = \Gamma k_{t-1} + d_t$$

$$k_t = \Delta_k k_{t-1} + \Theta_k i_t$$

$$h_t = \Delta_h h_{t-1} + \Theta_h c_t$$

$$s_t = \Lambda h_{t-1} + \Pi c_t$$

and

$$z_{t+1} = A_{22} z_t + C_2 w_{t+1}$$

$$b_t = U_b z_t$$

$$d_t = U_d z_t$$

The DLE class in Python maps this planning problem into a linear-quadratic dynamic programming problem and then solves it by using QuantEcon's LQ class.

(See Section 5.5 of Hansen & Sargent (2013) [Hansen and Sargent, 2013] for a full description of how to map these economies into an LQ setting, and how to use the solution to the LQ problem to construct the output matrices in order to simulate the economies)

The state for the LQ problem is

$$x_t = \begin{bmatrix} h_{t-1} \\ k_{t-1} \\ z_t \end{bmatrix}$$

and the control variable is $u_t = i_t$.

Once the LQ problem has been solved, the law of motion for the state is

$$x_{t+1} = (A - BF)x_t + Cw_{t+1}$$

where the optimal control law is $u_t = -Fx_t$.

Letting $A^o = A - BF$ we write this law of motion as

$$x_{t+1} = A^o x_t + Cw_{t+1}$$

20.3 Example Economies

Each of the example economies shown here will share a number of components. In particular, for each we will consider preferences of the form

$$-\frac{1}{2}\mathbb{E}\sum_{t=0}^{\infty}\beta^t[(s_t - b_t)^2 + l_t^2], 0 < \beta < 1$$

$$s_t = \lambda h_{t-1} + \pi c_t$$

$$h_t = \delta_h h_{t-1} + \theta_h c_t$$

$$b_t = U_b z_t$$

Technology of the form

$$c_t + i_t = \gamma_1 k_{t-1} + d_{1t}$$

$$k_t = \delta_k k_{t-1} + i_t$$

$$g_t = \phi_1 i_t \, , \phi_1 > 0$$

$$\begin{bmatrix} d_{1t} \\ 0 \end{bmatrix} = U_d z_t$$

And information of the form

$$z_{t+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} z_t + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} w_{t+1}$$

$$U_b = \begin{bmatrix} 30 & 0 & 0 \end{bmatrix}$$

$$U_d = \begin{bmatrix} 5 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We shall vary $\{\lambda, \pi, \delta_h, \theta_h, \gamma_1, \delta_k, \phi_1\}$ and the initial state x_0 across the three economies.

20.3.1 Example 1: Hall (1978)

First, we set parameters such that consumption follows a random walk. In particular, we set

$$\lambda = 0, \pi = 1, \gamma_1 = 0.1, \phi_1 = 0.00001, \delta_k = 0.95, \beta = \frac{1}{1.05}$$

(In this economy δ_h and θ_h are arbitrary as household capital does not enter the equation for consumption services We set them to values that will become useful in Example 3)

It is worth noting that this choice of parameter values ensures that $\beta(\gamma_1 + \delta_k) = 1$.

For simulations of this economy, we choose an initial condition of

$$x_0 = \begin{bmatrix} 5 & 150 & 1 & 0 & 0 \end{bmatrix}'$$

```
# Parameter Matrices
γ_1 = 0.1
ϕ_1 = 1e-5

ϕ_c, ϕ_g, ϕ_i, γ, δ_k, θ_k = (np.array([[1], [0]]),
                              np.array([[0], [1]]),
                              np.array([[1], [-ϕ_1]]),
                              np.array([[γ_1], [0]]),
                              np.array([[.95]]),
                              np.array([[1]]))

β, l_λ, π_h, δ_h, θ_h = (np.array([[1 / 1.05]]),
                         np.array([[0]]),
                         np.array([[1]]),
                         np.array([[.9]]),
                         np.array([[1]]) - np.array([[.9]]))

a22, c2, ub, ud = (np.array([[1,   0,   0],
                             [0, 0.8,   0],
                             [0,   0, 0.5]]),
                   np.array([[0, 0],
                             [1, 0],
                             [0, 1]]),
                   np.array([[30, 0, 0]]),
                   np.array([[5, 1, 0],
                             [0, 0, 0]]))

# Initial condition
x0 = np.array([[5], [150], [1], [0], [0]])

info1 = (a22, c2, ub, ud)
tech1 = (ϕ_c, ϕ_g, ϕ_i, γ, δ_k, θ_k)
pref1 = (β, l_λ, π_h, δ_h, θ_h)
```

These parameter values are used to define an economy of the DLE class.

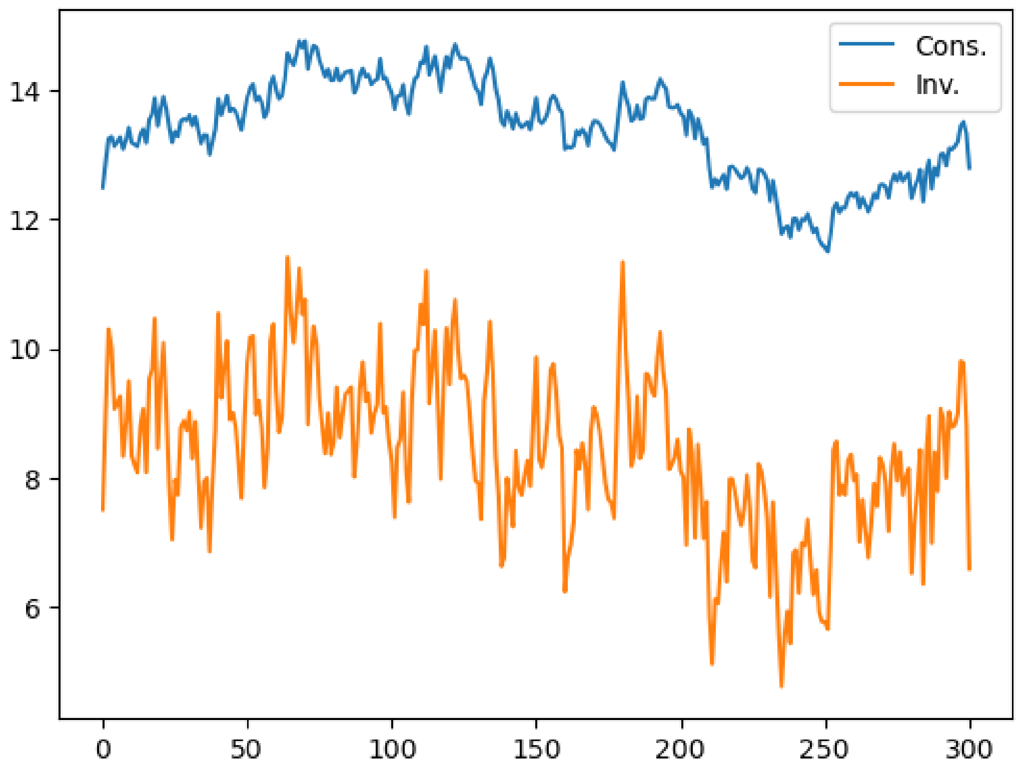
```
econ1 = DLE(info1, tech1, pref1)
```

We can then simulate the economy for a chosen length of time, from our initial state vector x_0

```
econ1.compute_sequence(x0, ts_length=300)
```

The economy stores the simulated values for each variable. Below we plot consumption and investment

```
# This is the right panel of Fig 5.7.1 from p.105 of HS2013
plt.plot(econ1.c[0], label='Cons.')
plt.plot(econ1.i[0], label='Inv.')
plt.legend()
plt.show()
```



Inspection of the plot shows that the sample paths of consumption and investment drift in ways that suggest that each has or nearly has a **random walk** or **unit root** component.

This is confirmed by checking the eigenvalues of A^o

```
econ1.endo, econ1.exo
```

```
(array([0.9, 1. ]), array([1. , 0.8, 0.5]))
```

The endogenous eigenvalue that appears to be unity reflects the random walk character of consumption in Hall's model.

- Actually, the largest endogenous eigenvalue is very slightly below 1.
- This outcome comes from the small adjustment cost ϕ_1.

```
econ1.endo[1]
```

```
0.9999999999904767
```

The fact that the largest endogenous eigenvalue is strictly less than unity in modulus means that it is possible to compute the non-stochastic steady state of consumption, investment and capital.

```
econ1.compute_steadystate()
np.set_printoptions(precision=3, suppress=True)
print(econ1.css, econ1.iss, econ1.kss)
```

```
[[4.999]] [[-0.001]] [[-0.023]]
```

However, the near-unity endogenous eigenvalue means that these steady state values are of little relevance.

20.3.2 Example 2: Altered Growth Condition

We generate our next economy by making two alterations to the parameters of Example 1.

- First, we raise ϕ_1 from 0.00001 to 1.
  - This will lower the endogenous eigenvalue that is close to 1, causing the economy to head more quickly to the vicinity of its non-stochastic steady-state.
- Second, we raise γ_1 from 0.1 to 0.15.
  - This has the effect of raising the optimal steady-state value of capital.

We also start the economy off from an initial condition with a lower capital stock

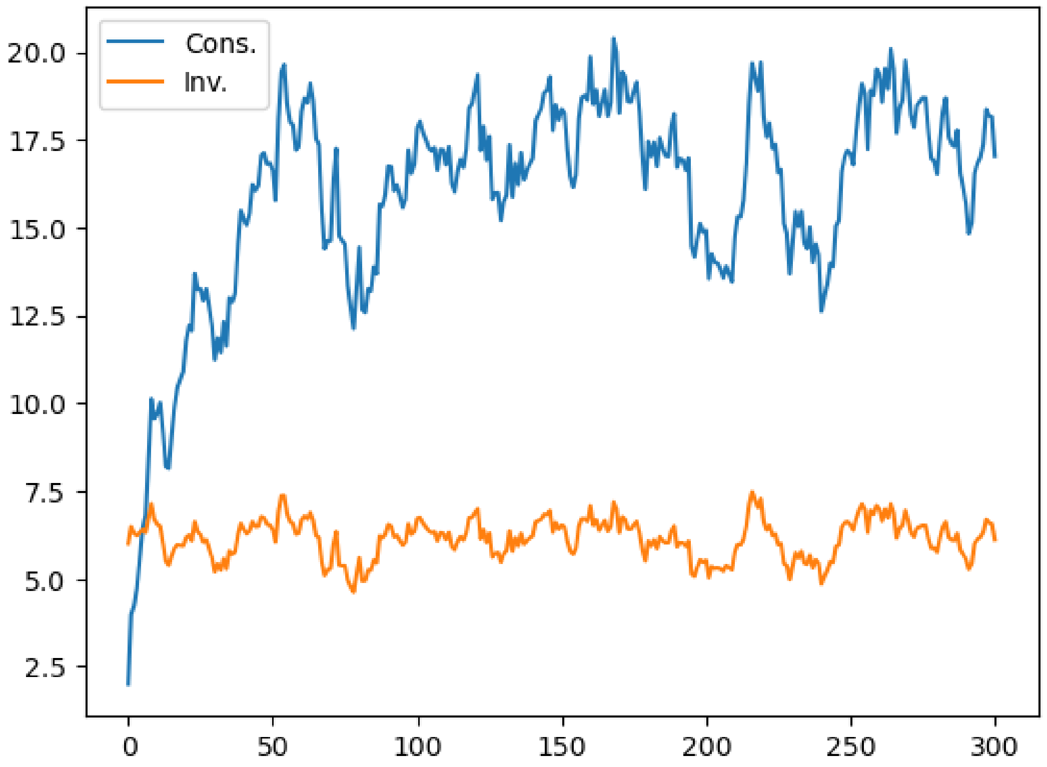
$$x_0 = \begin{bmatrix} 5 & 20 & 1 & 0 & 0 \end{bmatrix}'$$

Therefore, we need to define the following new parameters

```
γ2 = 0.15
γ22 = np.array([[γ2], [0]])

ϕ_12 = 1
ϕ_i2 = np.array([[1], [-ϕ_12]])

tech2 = (ϕ_c, ϕ_g, ϕ_i2, γ22, δ_k, θ_k)

x02 = np.array([[5], [20], [1], [0], [0]])
```

Creating the DLE class and then simulating gives the following plot for consumption and investment

```
econ2 = DLE(info1, tech2, pref1)

econ2.compute_sequence(x02, ts_length=300)

plt.plot(econ2.c[0], label='Cons.')
plt.plot(econ2.i[0], label='Inv.')
plt.legend()
plt.show()
```



Simulating our new economy shows that consumption grows quickly in the early stages of the sample.

However, it then settles down around the new non-stochastic steady-state level of consumption of 17.5, which we find as follows

```
econ2.compute_steadystate()
print(econ2.css, econ2.iss, econ2.kss)
```

```
[[17.5]] [[6.25]] [[125.]]
```

The economy converges faster to this level than in Example 1 because the largest endogenous eigenvalue of A^o is now significantly lower than 1.

```
econ2.endo, econ2.exo
```

```
(array([0.9  , 0.952]), array([1. , 0.8, 0.5]))
```

20.3.3 Example 3: A Jones-Manuelli (1990) Economy

For our third economy, we choose parameter values with the aim of generating *sustained* growth in consumption, investment and capital.

To do this, we set parameters so that Jones and Manuelli's "growth condition" is just satisfied.

In our notation, just satisfying the growth condition is actually equivalent to setting $\beta(\gamma_1 + \delta_k) = 1$, the condition that was necessary for consumption to be a random walk in Hall's model.

Thus, we lower γ_1 back to 0.1.

In our model, this is a necessary but not sufficient condition for growth.

To generate growth we set preference parameters to reflect habit persistence.

In particular, we set $\lambda = -1$, $\delta_h = 0.9$ and $\theta_h = 1 - \delta_h = 0.1$.

This makes preferences assume the form

$$-\frac{1}{2}\mathbb{E}\sum_{t=0}^{\infty}\beta^t[(c_t - b_t - (1-\delta_h)\sum_{j=0}^{\infty}\delta_h^j c_{t-j-1})^2 + l_t^2]$$

These preferences reflect habit persistence

- the effective "bliss point" $b_t + (1-\delta_h)\sum_{j=0}^{\infty}\delta_h^j c_{t-j-1}$ now shifts in response to a moving average of past consumption

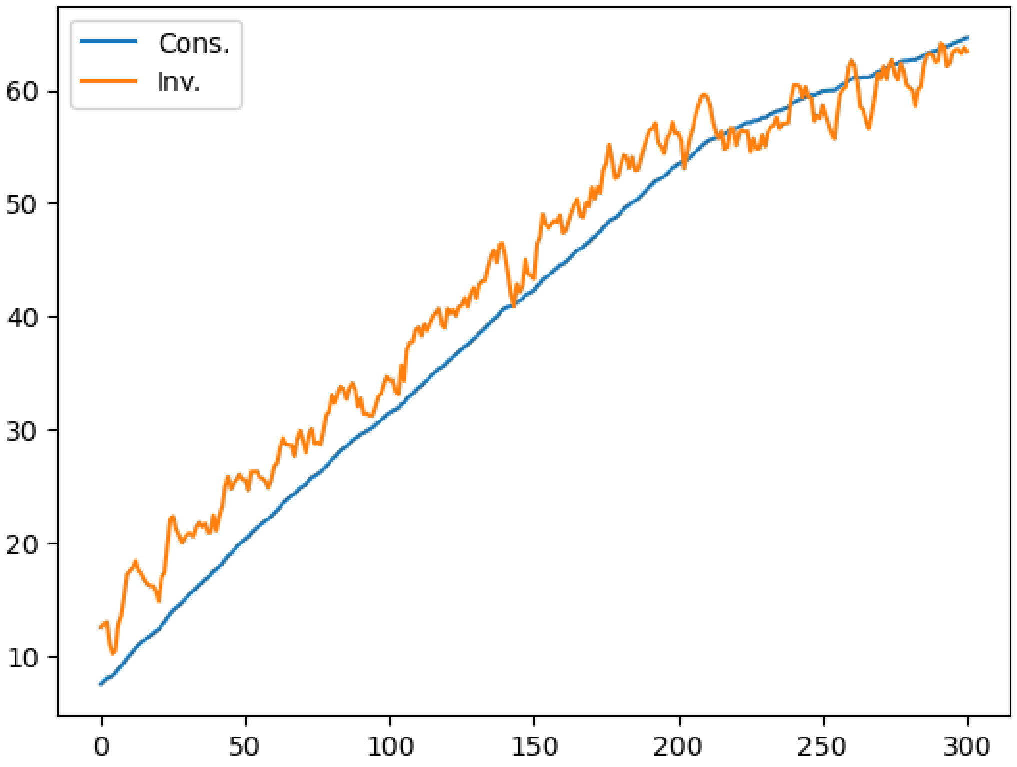
Since δ_h and θ_h were defined earlier, the only change we need to make from the parameters of Example 1 is to define the new value of λ.

```
l_λ2 = np.array([[-1]])
pref2 = (β, l_λ2, π_h, δ_h, θ_h)
```

```
econ3 = DLE(info1, tech1, pref2)
```

We simulate this economy from the original state vector

```
econ3.compute_sequence(x0, ts_length=300)

# This is the right panel of Fig 5.10.1 from p.110 of HS2013
plt.plot(econ3.c[0], label='Cons.')
plt.plot(econ3.i[0], label='Inv.')
plt.legend()
plt.show()
```



Thus, adding habit persistence to the Hall model of Example 1 is enough to generate sustained growth in our economy.

The eigenvalues of A^o in this new economy are

```
econ3.endo, econ3.exo
```

```
(array([1.+0.j, 1.-0.j]), array([1. , 0.8, 0.5]))
```

We now have two unit endogenous eigenvalues. One stems from satisfying the growth condition (as in Example 1).

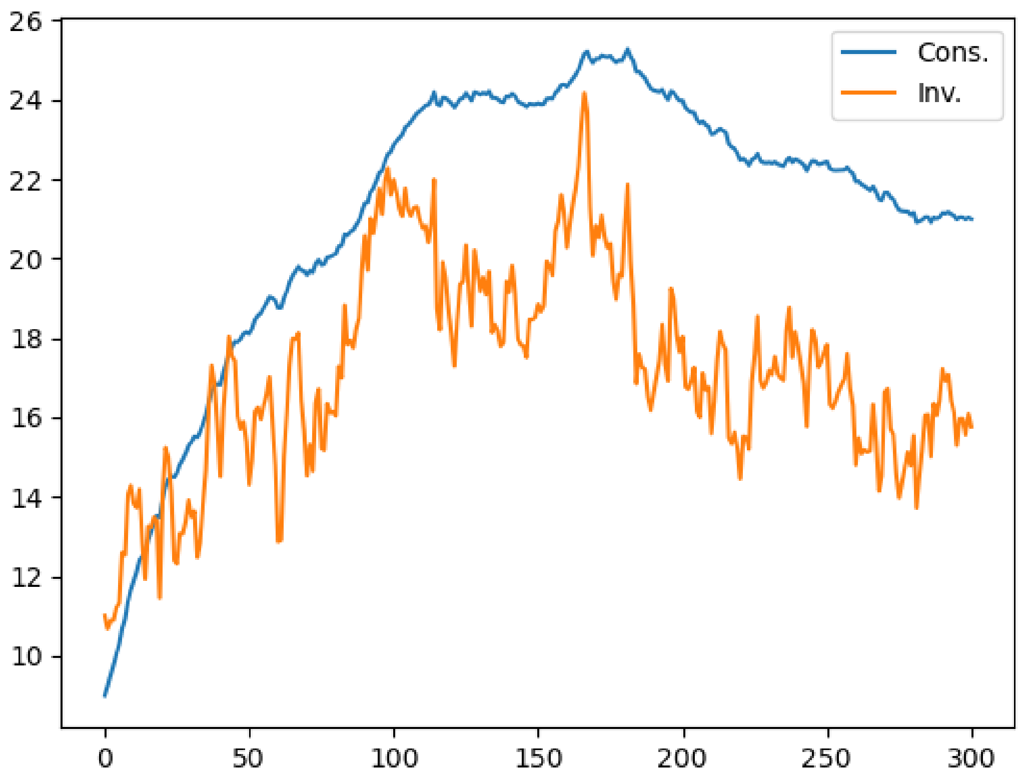
The other unit eigenvalue results from setting $\lambda = -1$.

To show the importance of both of these for generating growth, we consider the following experiments.

20.3.4 Example 3.1: Varying Sensitivity

Next we raise λ to -0.7

```
l_λ3 = np.array([[-0.7]])
pref3 = (β, l_λ3, π_h, δ_h, θ_h)

econ4 = DLE(info1, tech1, pref3)

econ4.compute_sequence(x0, ts_length=300)

plt.plot(econ4.c[0], label='Cons.')
plt.plot(econ4.i[0], label='Inv.')
plt.legend()
plt.show()
```



We no longer achieve sustained growth if λ is raised from -1 to -0.7.

This is related to the fact that one of the endogenous eigenvalues is now less than 1.

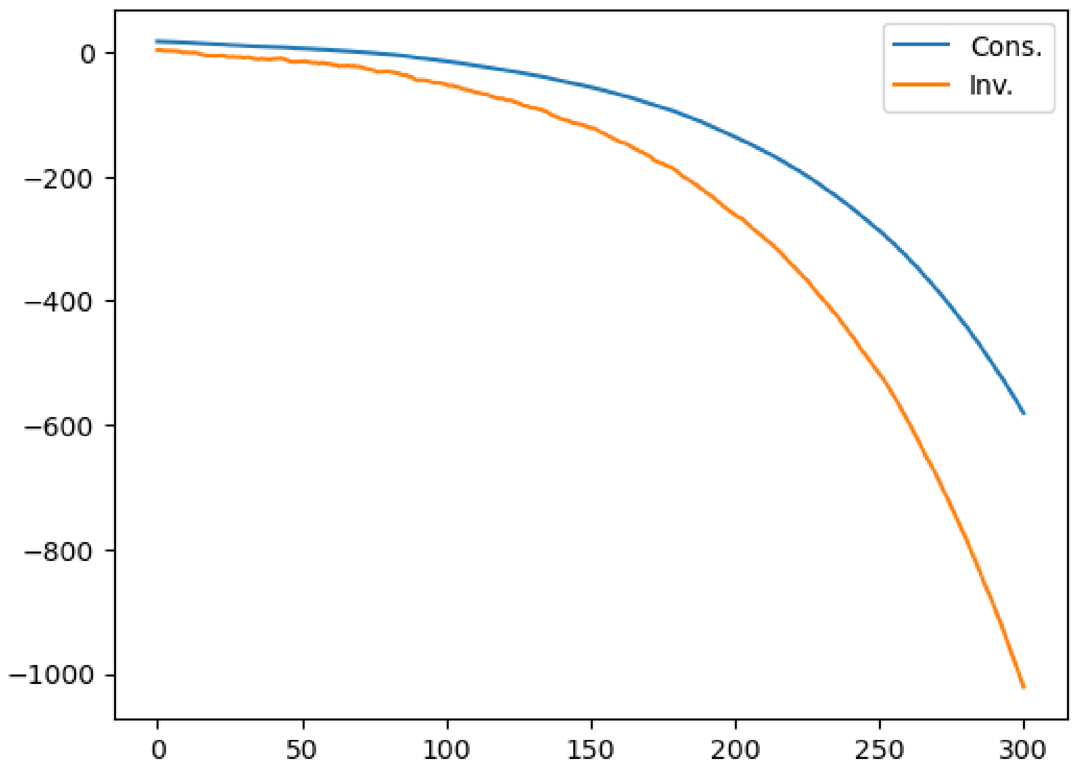
```
econ4.endo, econ4.exo
```

```
(array([0.97, 1.  ]), array([1. , 0.8, 0.5]))
```

20.3.5 Example 3.2: More Impatience

Next let's lower β to 0.94

```
β_2 = np.array([[0.94]])
pref4 = (β_2, l_λ, π_h, δ_h, θ_h)

econ5 = DLE(info1, tech1, pref4)

econ5.compute_sequence(x0, ts_length=300)

plt.plot(econ5.c[0], label='Cons.')
plt.plot(econ5.i[0], label='Inv.')
plt.legend()
plt.show()
```



Growth also fails if we lower β, since we now have $\beta(\gamma_1 + \delta_k) < 1$.

Consumption and investment explode downwards, as a lower value of β causes the representative consumer to front-load consumption.

This explosive path shows up in the second endogenous eigenvalue now being larger than one.

```
econ5.endo, econ5.exo
```

```
(array([0.9  , 1.013]), array([1. , 0.8, 0.5]))
```

CHAPTER

TWENTYONE

LUCAS ASSET PRICING USING DLE

Contents

This is one of a suite of lectures that use the quantecon DLE class to instantiate models within the [Hansen and Sargent, 2013] class of models described in detail in *Recursive Models of Dynamic Linear Economies*.

In addition to what's in Anaconda, this lecture uses the quantecon library

```
!pip install --upgrade quantecon
```

This lecture uses the DLE class to price payout streams that are linear functions of the economy's state vector, as well as risk-free assets that pay out one unit of the first consumption good with certainty.

We assume basic knowledge of the class of economic environments that fall within the domain of the DLE class.

Many details about the basic environment are contained in the lecture *Growth in Dynamic Linear Economies*.

We'll also need the following imports

```
import numpy as np
import matplotlib.pyplot as plt
from quantecon import DLE
```

We use a linear-quadratic version of an economy that Lucas (1978) [Lucas, 1978] used to develop an equilibrium theory of asset prices:

Preferences

$$-\frac{1}{2}\mathbb{E}\sum_{t=0}^{\infty}\beta^t[(c_t-b_t)^2+l_t^2]|J_0$$

$$s_t = c_t$$

$$b_t = U_b z_t$$

Technology

$$c_t = d_{1t}$$

$$k_t = \delta_k k_{t-1} + i_t$$

$$g_t = \phi_1 i_t \, , \phi_1 > 0$$

$$\begin{bmatrix} d_{1t} \\ 0 \end{bmatrix} = U_d z_t$$

Information

$$z_{t+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} z_t + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} w_{t+1}$$

$$U_b = \begin{bmatrix} 30 & 0 & 0 \end{bmatrix}$$

$$U_d = \begin{bmatrix} 5 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$x_0 = \begin{bmatrix} 5 & 150 & 1 & 0 & 0 \end{bmatrix}'$$

21.1 Asset Pricing Equations

[Hansen and Sargent, 2013] show that the time t value of a permanent claim to a stream $y_s = U_a x_s \, , s \geq t$ is:

$$a_t = (x_t' \mu_a x_t + \sigma_a)/(\bar{e}_1 M_c x_t)$$

with

$$\mu_a = \sum_{\tau=0}^{\infty} \beta^\tau (A^{o'})^\tau Z_a A^{o\tau}$$

$$\sigma_a = \frac{\beta}{1-\beta} \text{trace}(Z_a \sum_{\tau=0}^{\infty} \beta^\tau (A^o)^\tau C C' (A^{o'})^\tau)$$

where

$$Z_a = U_a' M_c$$

The use of $\bar{e}_1$ indicates that the first consumption good is the numeraire.

21.2 Asset Pricing Simulations

```
gam = 0
γ = np.array([[gam], [0]])
ϕ_c = np.array([[1], [0]])
ϕ_g = np.array([[0], [1]])
ϕ_1 = 1e-4
ϕ_i = np.array([[0], [-ϕ_1]])
δ_k = np.array([[.95]])
θ_k = np.array([[1]])
β = np.array([[1 / 1.05]])
ud = np.array([[5, 1, 0],
               [0, 0, 0]])
a22 = np.array([[1,   0,    0],
```

(continues on next page)

(continued from previous page)

```
                 [0, 0.8,    0],
                 [0,   0, 0.5]])
c2 = np.array([[0, 1, 0],
                [0, 0, 1]]).T
l_λ = np.array([[0]])
π_h = np.array([[1]])
δ_h = np.array([[.9]])
θ_h = np.array([[1]]) - δ_h
ub = np.array([[30, 0, 0]])
x0 = np.array([[5, 150, 1, 0, 0]]).T

info1 = (a22, c2, ub, ud)
tech1 = (ϕ_c, ϕ_g, ϕ_i, γ, δ_k, θ_k)
pref1 = (β, l_λ, π_h, δ_h, θ_h)
```

```
econ1 = DLE(info1, tech1, pref1)
```

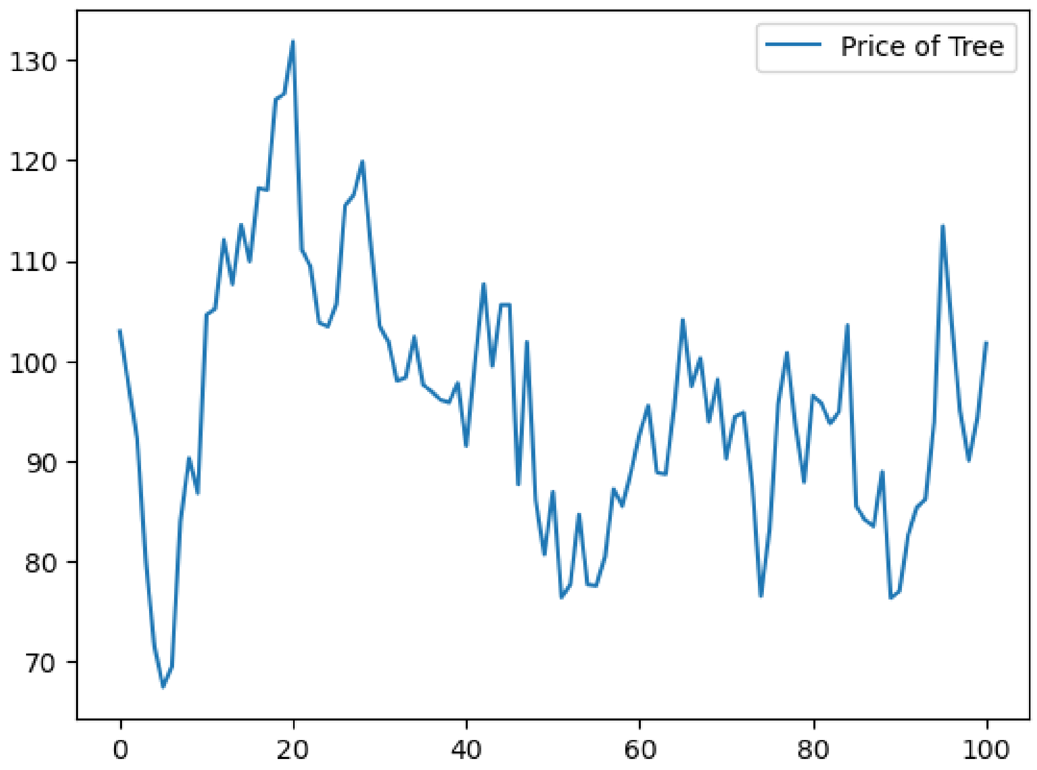
After specifying a "Pay" matrix, we simulate the economy.

The particular choice of "Pay" used below means that we are pricing a perpetual claim on the endowment process d_{1t}

```
econ1.compute_sequence(x0, ts_length=100, Pay=np.array([econ1.Sd[0, :]]))
```

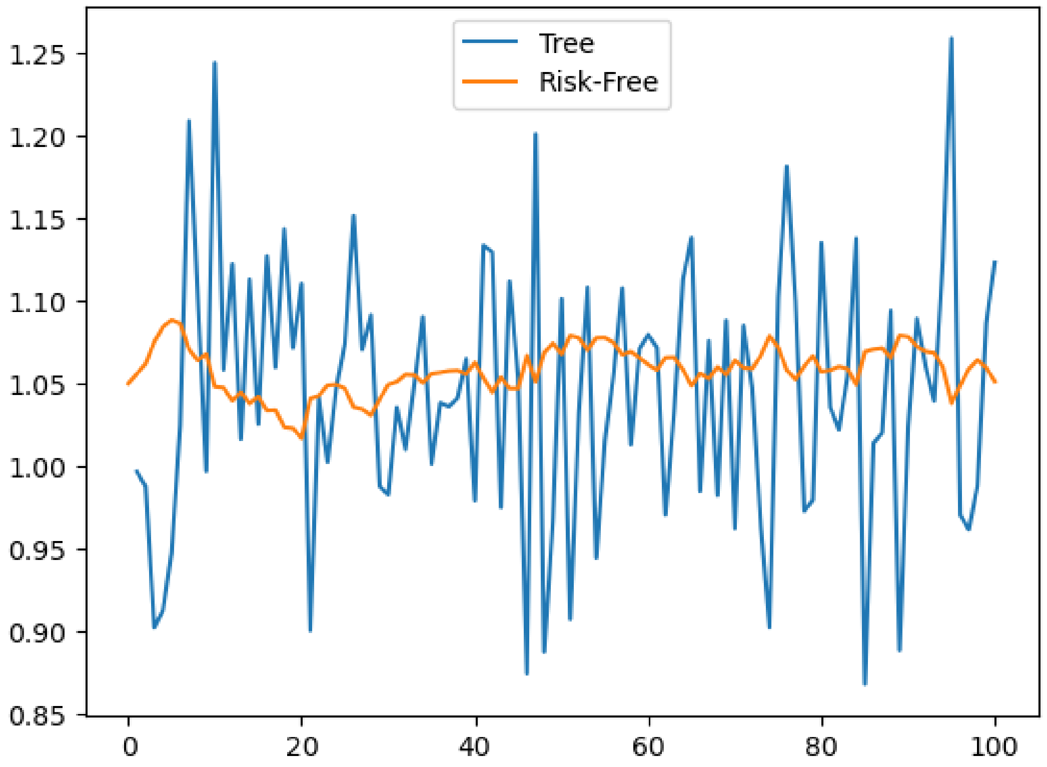
The graph below plots the price of this claim over time:

```
### Fig 7.12.1 from p.147 of HS2013
plt.plot(econ1.Pay_Price, label='Price of Tree')
plt.legend()
plt.show()
```



The next plot displays the realized gross rate of return on this “Lucas tree” as well as on a risk-free one-period bond:

```
### Left panel of Fig 7.12.2 from p.148 of HS2013
plt.plot(econ1.Pay_Gross, label='Tree')
plt.plot(econ1.R1_Gross, label='Risk-Free')
plt.legend()
plt.show()
```



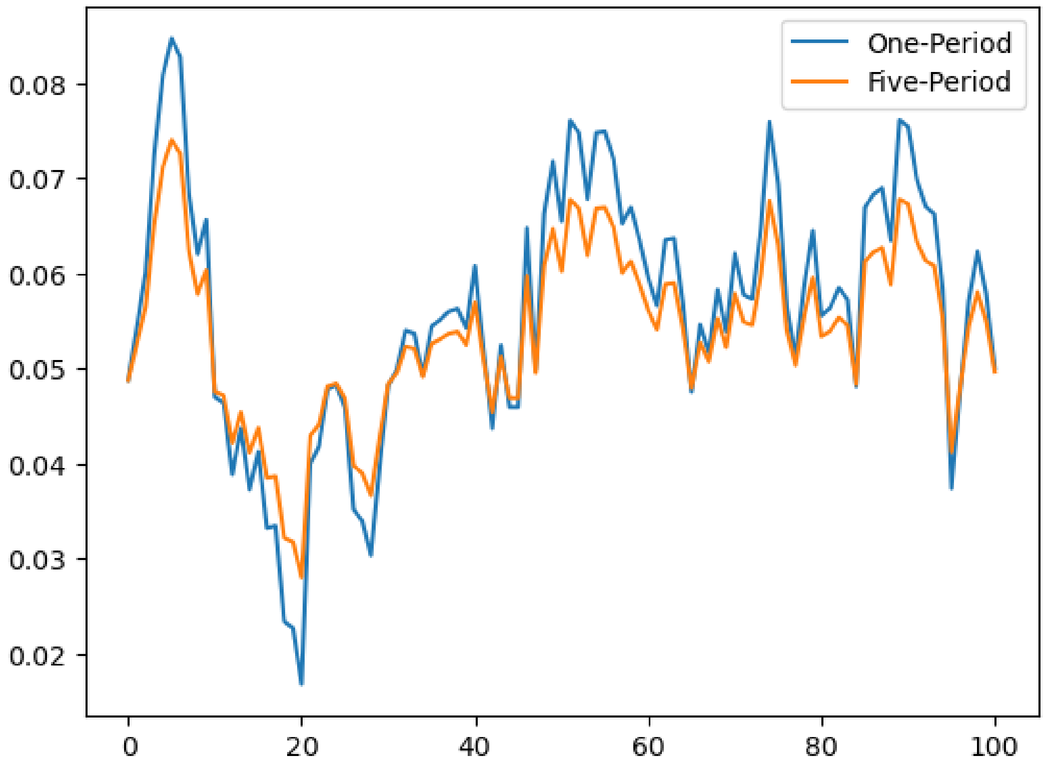
```
np.corrcoef(econ1.Pay_Gross[1:, 0], econ1.R1_Gross[1:, 0])
```

```
array([[ 1.        , -0.44842721],
       [-0.44842721,  1.        ]])
```

Above we have also calculated the correlation coefficient between these two returns.

To give an idea of how the term structure of interest rates moves in this economy, the next plot displays the *net* rates of return on one-period and five-period risk-free bonds:

```
### Right panel of Fig 7.12.2 from p.148 of HS2013
plt.plot(econ1.R1_Net, label='One-Period')
plt.plot(econ1.R5_Net, label='Five-Period')
plt.legend()
plt.show()
```



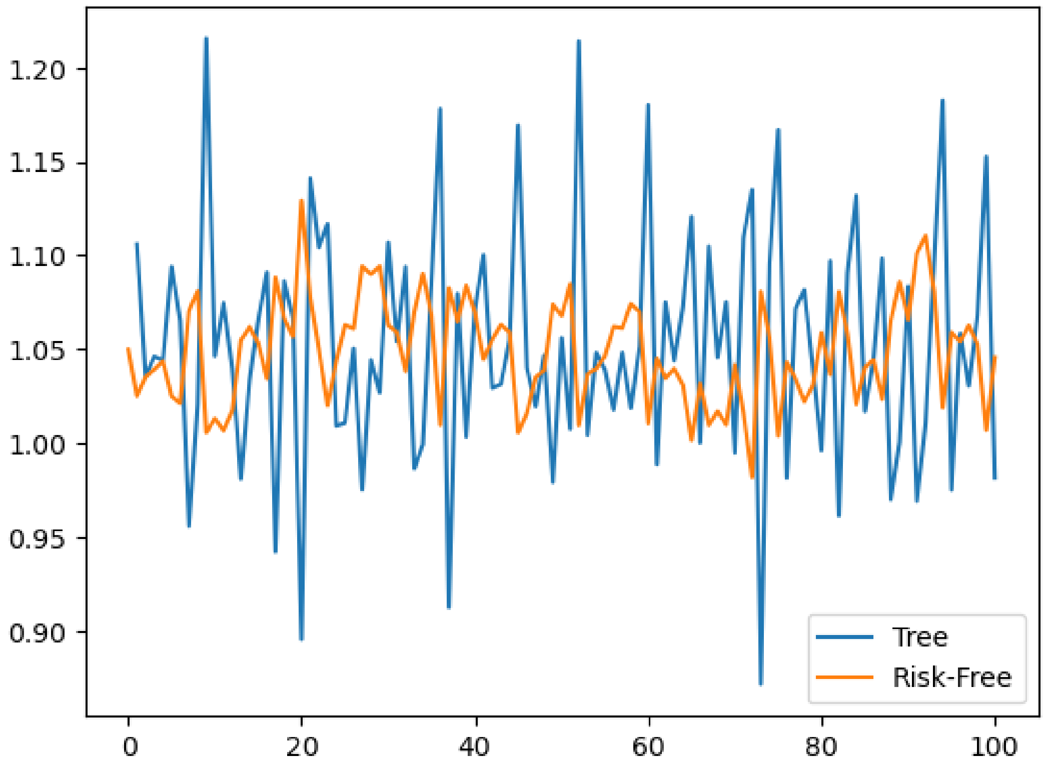
From the above plot, we can see the tendency of the term structure to slope up when rates are low and to slope down when rates are high.

Comparing it to the previous plot of the price of the "Lucas tree", we can also see that net rates of return are low when the price of the tree is high, and vice versa.

We now plot the realized gross rate of return on a "Lucas tree" as well as on a risk-free one-period bond when the autoregressive parameter for the endowment process is reduced to 0.4:

```
a22_2 = np.array([[1,   0,   0],
                  [0, 0.4,   0],
                  [0,   0, 0.5]])
info2 = (a22_2, c2, ub, ud)

econ2 = DLE(info2, tech1, pref1)
econ2.compute_sequence(x0, ts_length=100, Pay=np.array([econ2.Sd[0, :]]))
```

```
### Left panel of Fig 7.12.3 from p.148 of HS2013
plt.plot(econ2.Pay_Gross, label='Tree')
plt.plot(econ2.R1_Gross, label='Risk-Free')
plt.legend()
plt.show()
```



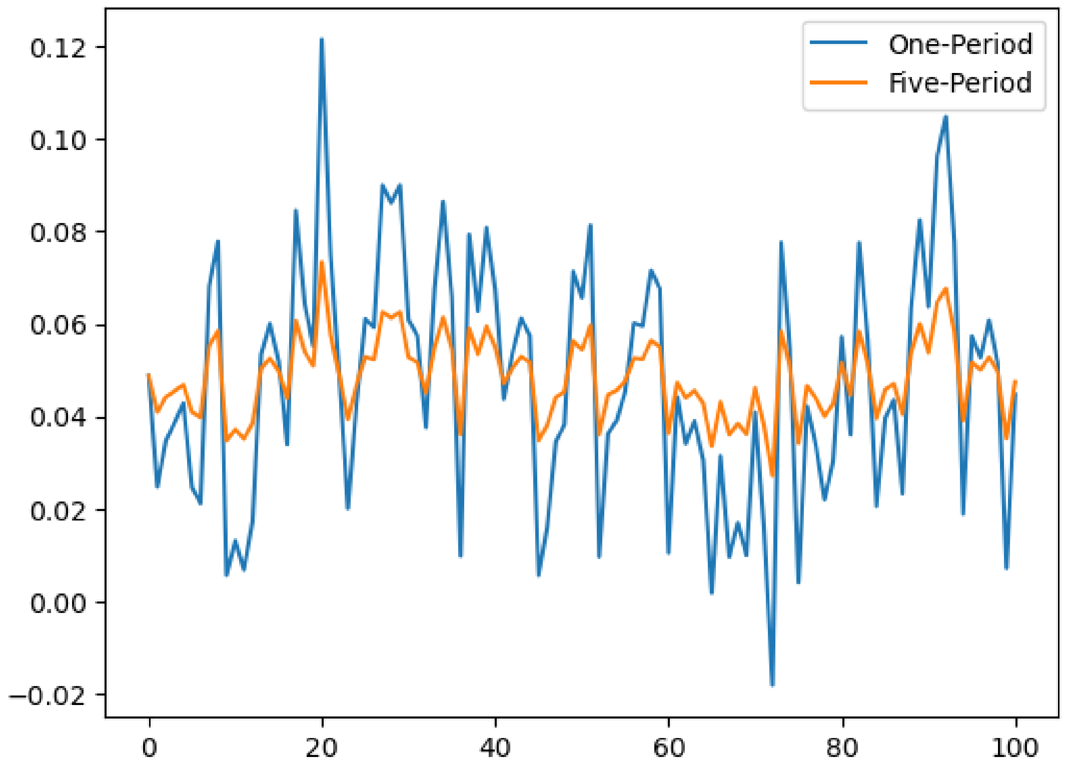
```
np.corrcoef(econ2.Pay_Gross[1:, 0], econ2.R1_Gross[1:, 0])
```

```
array([[ 1.        , -0.65282644],
       [-0.65282644,  1.        ]])
```

The correlation between these two gross rates is now more negative.

Next, we again plot the *net* rates of return on one-period and five-period risk-free bonds:

```
### Right panel of Fig 7.12.3 from p.148 of HS2013
plt.plot(econ2.R1_Net, label='One-Period')
plt.plot(econ2.R5_Net, label='Five-Period')
plt.legend()
plt.show()
```



We can see the tendency of the term structure to slope up when rates are low (and down when rates are high) has been accentuated relative to the first instance of our economy.

CHAPTER

TWENTYTWO

IRFS IN HALL MODELS

Contents

This is another member of a suite of lectures that use the quantecon DLE class to instantiate models within the [Hansen and Sargent, 2013] class of models described in detail in *Recursive Models of Dynamic Linear Economies*.

In addition to what's in Anaconda, this lecture uses the quantecon library.

```
!pip install --upgrade quantecon
```

We'll make these imports:

```
import numpy as np
import matplotlib.pyplot as plt
from quantecon import DLE
```

This lecture shows how the DLE class can be used to create impulse response functions for three related economies, starting from Hall (1978) [Hall, 1978].

Knowledge of the basic economic environment is assumed.

See the lecture "Growth in Dynamic Linear Economies" for more details.

22.1 Example 1: Hall (1978)

First, we set parameters to make consumption (almost) follow a random walk.

We set

$$\lambda = 0, \pi = 1, \gamma_1 = 0.1, \phi_1 = 0.00001, \delta_k = 0.95, \beta = \frac{1}{1.05}$$

(In this example δ_h and θ_h are arbitrary as household capital does not enter the equation for consumption services.

We set them to values that will become useful in Example 3)

It is worth noting that this choice of parameter values ensures that $\beta(\gamma_1 + \delta_k) = 1$.

For simulations of this economy, we choose an initial condition of:

$$x_0 = \begin{bmatrix} 5 & 150 & 1 & 0 & 0 \end{bmatrix}'$$

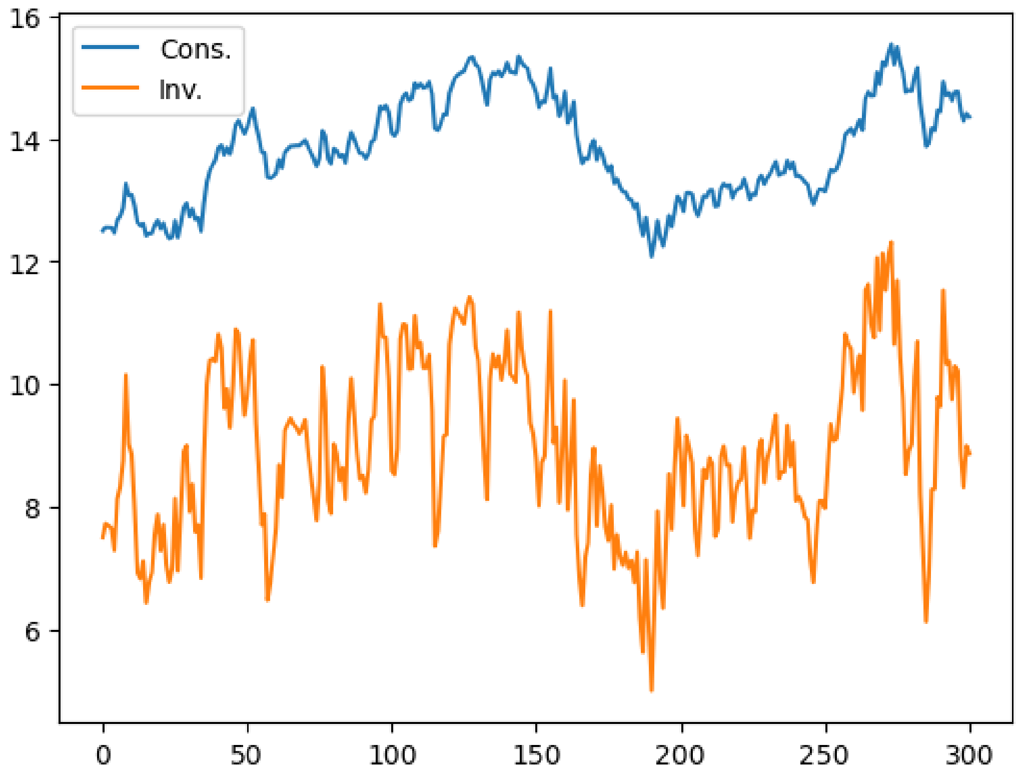
```
γ_1 = 0.1
γ = np.array([[γ_1], [0]])
ϕ_c = np.array([[1], [0]])
ϕ_g = np.array([[0], [1]])
ϕ_1 = 1e-5
ϕ_i = np.array([[1], [-ϕ_1]])
δ_k = np.array([[.95]])
θ_k = np.array([[1]])
β = np.array([[1 / 1.05]])
l_λ = np.array([[0]])
π_h = np.array([[1]])
δ_h = np.array([[.9]])
θ_h = np.array([[1]])
a22 = np.array([[1,   0,   0],
                [0, 0.8,   0],
                [0,   0, 0.5]])
c2 = np.array([[0, 0],
               [1, 0],
               [0, 1]])
ud = np.array([[5, 1, 0],
               [0, 0, 0]])
ub = np.array([[30, 0, 0]])
x0 = np.array([[5], [150], [1], [0], [0]])

info1 = (a22, c2, ub, ud)
tech1 = (ϕ_c, ϕ_g, ϕ_i, γ, δ_k, θ_k)
pref1 = (β, l_λ, π_h, δ_h, θ_h)
```

These parameter values are used to define an economy of the DLE class.

We can then simulate the economy for a chosen length of time, from our initial state vector x_0.

The economy stores the simulated values for each variable. Below we plot consumption and investment:

```
econ1 = DLE(info1, tech1, pref1)
econ1.compute_sequence(x0, ts_length=300)

# This is the right panel of Fig 5.7.1 from p.105 of HS2013
plt.plot(econ1.c[0], label='Cons.')
plt.plot(econ1.i[0], label='Inv.')
plt.legend()
plt.show()
```

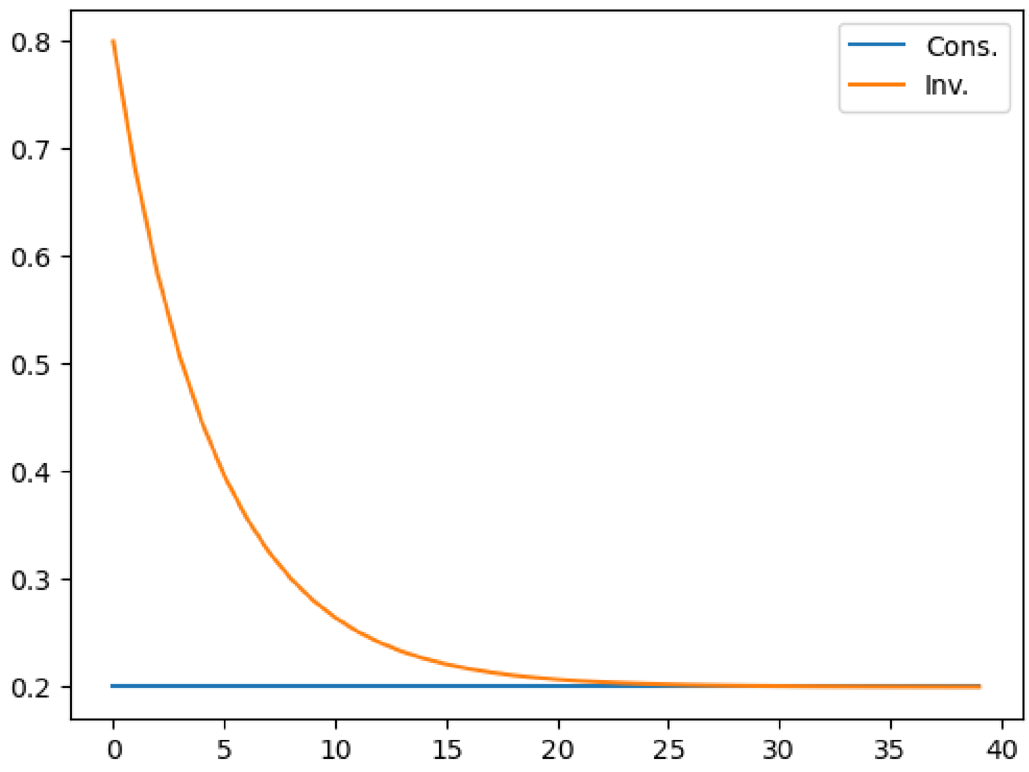


The DLE class can be used to create impulse response functions for each of the endogenous variables: $\{c_t, s_t, h_t, i_t, k_t, g_t\}$.

If no selector vector for the shock is specified, the default choice is to give IRFs to the first shock in w_{t+1}.

Below we plot the impulse response functions of investment and consumption to an endowment innovation (the first shock) in the Hall model:

```
econ1.irf(ts_length=40, shock=None)
# This is the left panel of Fig 5.7.1 from p.105 of HS2013
plt.plot(econ1.c_irf, label='Cons.')
plt.plot(econ1.i_irf, label='Inv.')
plt.legend()
plt.show()
```



It can be seen that the endowment shock has permanent effects on the level of both consumption and investment, consistent with the endogenous unit eigenvalue in this economy.

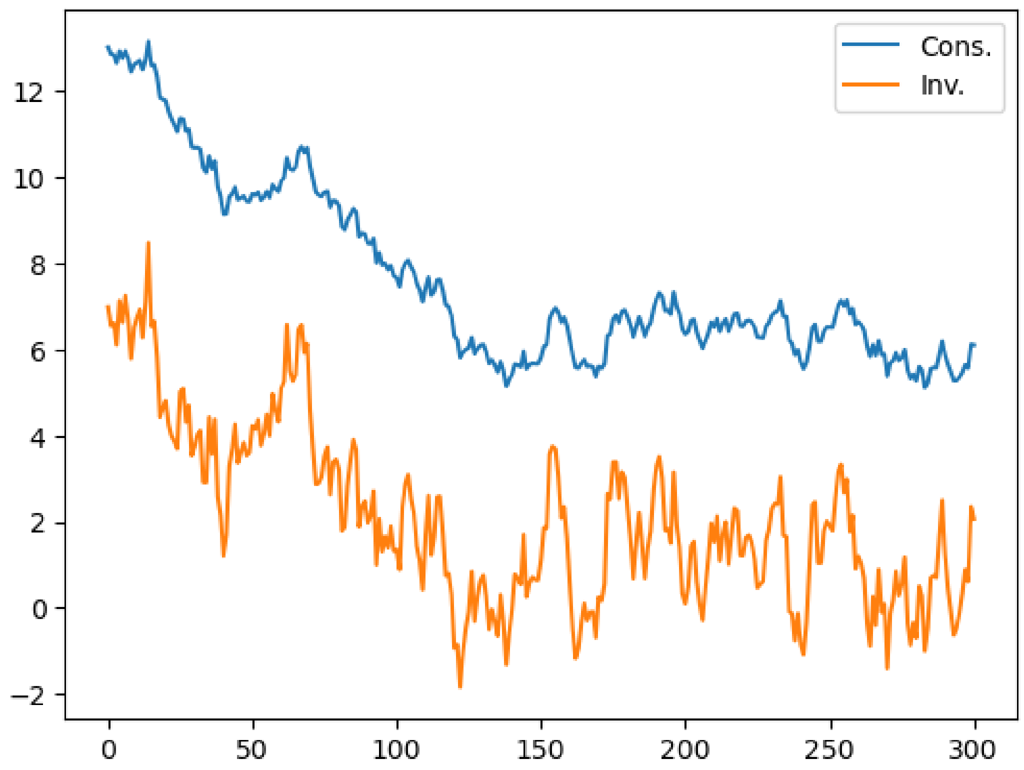
Investment is much more responsive to the endowment shock at shorter time horizons.

22.2 Example 2: Higher Adjustment Costs

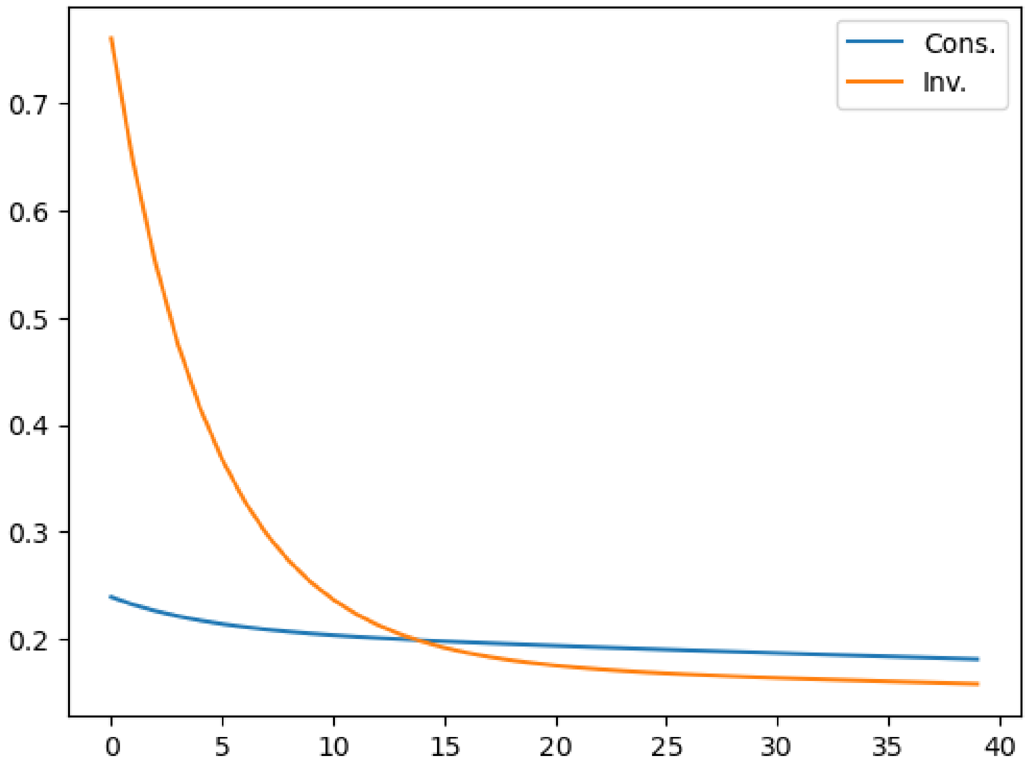
We generate our next economy by making only one change to the parameters of Example 1: we raise the parameter associated with the cost of adjusting capital,ϕ_1, from 0.00001 to 0.2.

This will lower the endogenous eigenvalue that is unity in Example 1 to a value slightly below 1.

```
ϕ_12 = 0.2
ϕ_i2 = np.array([[1], [-ϕ_12]])
tech2 = (ϕ_c, ϕ_g, ϕ_i2, γ, δ_k, θ_k)

econ2 = DLE(info1, tech2, pref1)
econ2.compute_sequence(x0, ts_length = 300)

# This is the right panel of Fig 5.8.1 from p.106 of HS2013
plt.plot(econ2.c[0], label='Cons.')
plt.plot(econ2.i[0], label='Inv.')
plt.legend()
plt.show()
```



```
econ2.irf(ts_length=40,shock=None)
# This is the left panel of Fig 5.8.1 from p.106 of HS2013
plt.plot(econ2.c_irf,label='Cons.')
plt.plot(econ2.i_irf,label='Inv.')
plt.legend()
plt.show()
```



```
econ2.endo
```

```
array([0.9       , 0.99657126])
```

```
econ2.compute_steadystate()
print(econ2.css, econ2.iss, econ2.kss)
```

```
[[5.]] [[2.92940472e-12]] [[5.85879555e-11]]
```

The first graph shows that there seems to be a downward trend in both consumption and investment.

This is a consequence of the decrease in the largest endogenous eigenvalue from unity in the earlier economy, caused by the higher adjustment cost.

The present economy has a nonstochastic steady state value of 5 for consumption and 0 for both capital and investment.

Because the largest endogenous eigenvalue is still close to 1, the economy heads only slowly towards these mean values.

The impulse response functions now show that an endowment shock does not have a permanent effect on the levels of either consumption or investment.

22.3 Example 3: Durable Consumption Goods

We generate our third economy by raising ϕ_1 further, to 1.0. We also raise the production function parameter from 0.1 to 0.15 (which raises the non-stochastic steady state value of capital above zero).

We also change the specification of preferences to make the consumption good *durable*.

Specifically, we allow for a single durable household good obeying:

$$h_t = \delta_h h_{t-1} + c_t \, , 0 < \delta_h < 1$$

Services are related to the stock of durables at the beginning of the period:

$$s_t = \lambda h_{t-1} \, , \lambda > 0$$

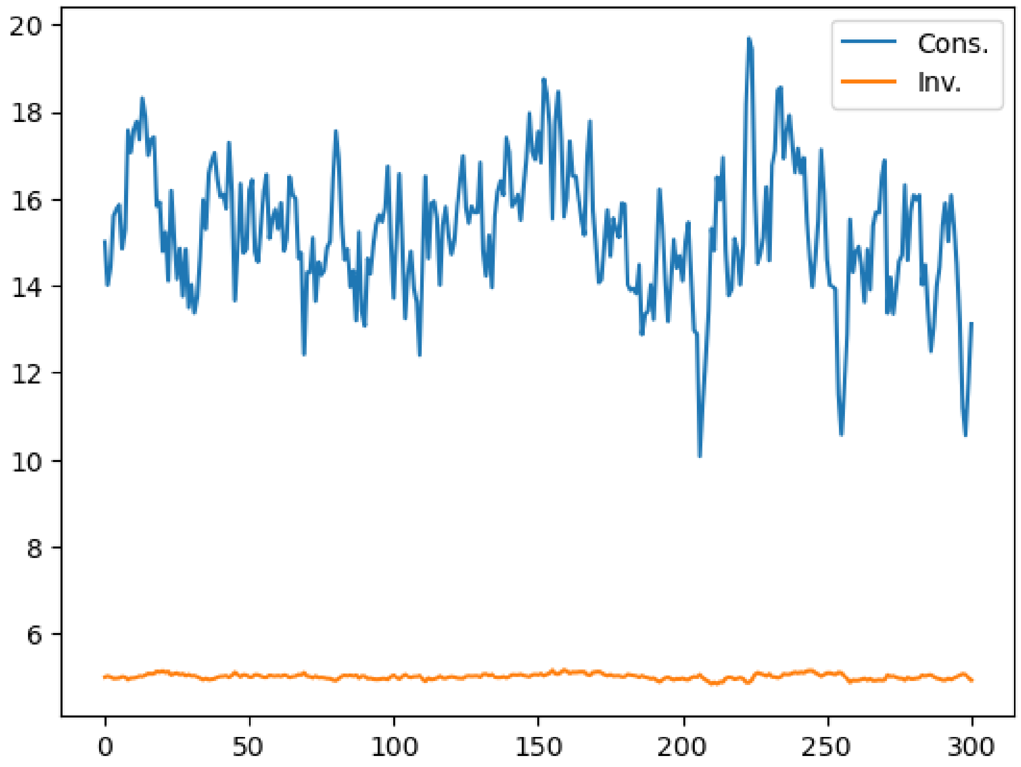
And preferences are ordered by:

$$-\frac{1}{2}\mathbb{E}\sum_{t=0}^{\infty}\beta^t[(\lambda h_{t-1} - b_t)^2 + l_t^2]|J_0$$

To implement this, we set $\lambda = 0.1$ and $\pi = 0$ (we have already set $\theta_h = 1$ and $\delta_h = 0.9$).

We start from an initial condition that makes consumption begin near around its non-stochastic steady state.

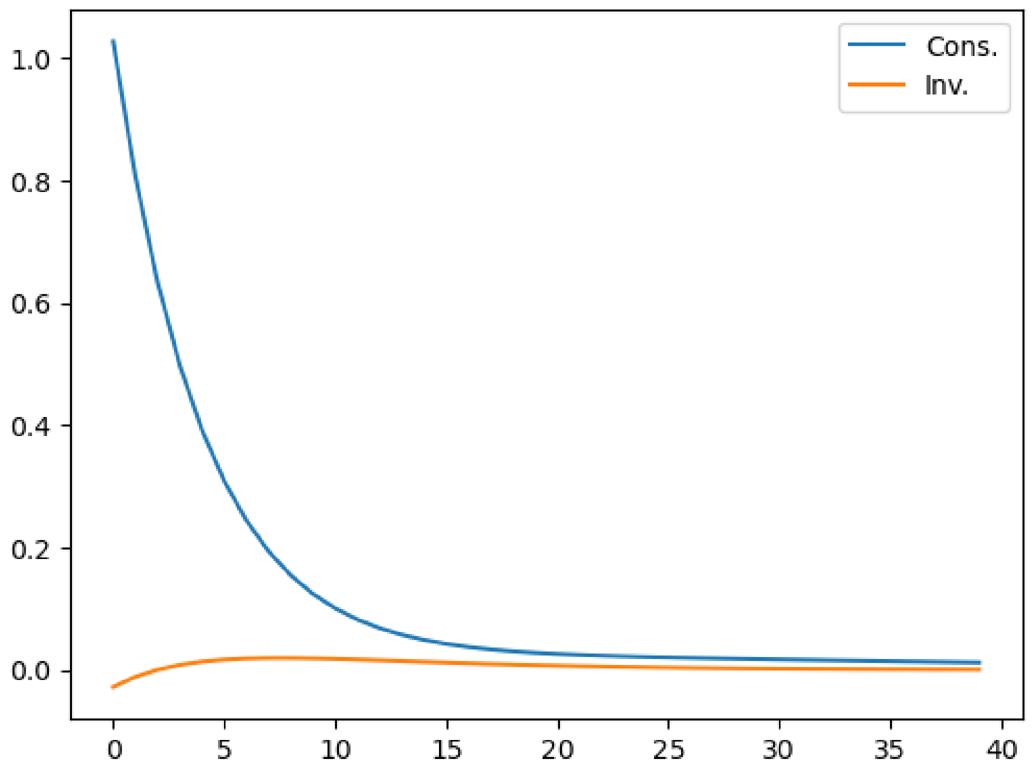
```
ϕ_13 = 1
ϕ_i3 = np.array([[1], [-ϕ_13]])

γ_12 = 0.15
γ_2 = np.array([[γ_12], [0]])

l_λ2 = np.array([[0.1]])
π_h2 = np.array([[0]])

x01 = np.array([[150], [100], [1], [0], [0]])

tech3 = (ϕ_c, ϕ_g, ϕ_i3, γ_2, δ_k, θ_k)
pref2 = (β, l_λ2, π_h2, δ_h, θ_h)

econ3 = DLE(info1, tech3, pref2)
econ3.compute_sequence(x01, ts_length=300)

# This is the right panel of Fig 5.11.1 from p.111 of HS2013
plt.plot(econ3.c[0], label='Cons.')
plt.plot(econ3.i[0], label='Inv.')
plt.legend()
plt.show()
```



In contrast to Hall's original model of Example 1, it is now investment that is much smoother than consumption.

This illustrates how making consumption goods durable tends to undo the strong consumption smoothing result that Hall obtained.

```
econ3.irf(ts_length=40, shock=None)
# This is the left panel of Fig 5.11.1 from p.111 of HS2013
plt.plot(econ3.c_irf, label='Cons.')
plt.plot(econ3.i_irf, label='Inv.')
plt.legend()
plt.show()
```



The impulse response functions confirm that consumption is now much more responsive to an endowment shock (and investment less so) than in Example 1.

As in Example 2, the endowment shock has permanent effects on neither variable.

CHAPTER

TWENTYTHREE

PERMANENT INCOME MODEL USING THE DLE CLASS

Contents

This lecture is part of a suite of lectures that use the quantecon DLE class to instantiate models within the [Hansen and Sargent, 2013] class of models described in detail in *Recursive Models of Dynamic Linear Economies*.

In addition to what's included in Anaconda, this lecture uses the quantecon library.

```
!pip install --upgrade quantecon
```

This lecture adds a third solution method for the linear-quadratic-Gaussian permanent income model with $\beta R = 1$, complementing the other two solution methods described in Optimal Savings I: The Permanent Income Model and Optimal Savings II: LQ Techniques and this Jupyter notebook.

The additional solution method uses the **DLE** class.

In this way, we map the permanent income model into the framework of Hansen & Sargent (2013) "Recursive Models of Dynamic Linear Economies" [Hansen and Sargent, 2013].

We'll also require the following imports

```
import numpy as np
import matplotlib.pyplot as plt
from quantecon import DLE

np.set_printoptions(suppress=True, precision=4)
```

23.1 The Permanent Income Model

The LQ permanent income model is an example of a **savings problem**.

A consumer has preferences over consumption streams that are ordered by the utility functional

$$E_0 \sum_{t=0}^{\infty} \beta^t u(c_t) \tag{23.1}$$

where E_t is the mathematical expectation conditioned on the consumer's time t information, c_t is time t consumption, $u(c)$ is a strictly concave one-period utility function, and $\beta \in (0,1)$ is a discount factor.

The LQ model gets its name partly from assuming that the utility function u is quadratic:

$$u(c) = -.5(c - \gamma)^2$$

where $\gamma > 0$ is a bliss level of consumption.

The consumer maximizes the utility functional (23.1) by choosing a consumption, borrowing plan $\{c_t, b_{t+1}\}_{t=0}^{\infty}$ subject to the sequence of budget constraints

$$c_t + b_t = R^{-1} b_{t+1} + y_t, t \geq 0 \tag{23.2}$$

where y_t is an exogenous stationary endowment process, R is a constant gross risk-free interest rate, b_t is one-period risk-free debt maturing at t, and b_0 is a given initial condition.

We shall assume that $R^{-1} = \beta$.

Equation (23.2) is linear.

We use another set of linear equations to model the endowment process.

In particular, we assume that the endowment process has the state-space representation

$$\begin{aligned} z_{t+1} &= A_{22} z_t + C_2 w_{t+1} \\ y_t &= U_y z_t \end{aligned} \tag{23.3}$$

where w_{t+1} is an IID process with mean zero and identity contemporaneous covariance matrix, A_{22} is a stable matrix, its eigenvalues being strictly below unity in modulus, and U_y is a selection vector that identifies y with a particular linear combination of the z_t.

We impose the following condition on the consumption, borrowing plan:

$$E_0 \sum_{t=0}^{\infty} \beta^t b_t^2 < +\infty \tag{23.4}$$

This condition suffices to rule out Ponzi schemes.

(We impose this condition to rule out a borrow-more-and-more plan that would allow the household to enjoy bliss consumption forever)

The state vector confronting the household at t is

$$x_t = \begin{bmatrix} z_t \\ b_t \end{bmatrix}$$

where b_t is its one-period debt falling due at the beginning of period t and z_t contains all variables useful for forecasting its future endowment.

We assume that $\{y_t\}$ follows a second order univariate autoregressive process:

$$y_{t+1} = \alpha + \rho_1 y_t + \rho_2 y_{t-1} + \sigma w_{t+1}$$

23.1.1 Solution with the DLE Class

One way of solving this model is to map the problem into the framework outlined in Section 4.8 of [Hansen and Sargent, 2013] by setting up our technology, information and preference matrices as follows:

Technology: $\phi_c = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \phi_g = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \phi_i = \begin{bmatrix} -1 \\ -0.00001 \end{bmatrix}, \Gamma = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \Delta_k = 0, \ \Theta_k = R.$

Information: $A_{22} = \begin{bmatrix} 1 & 0 & 0 \\ \alpha & \rho_1 & \rho_2 \\ 0 & 1 & 0 \end{bmatrix}, C_2 = \begin{bmatrix} 0 \\ \sigma \\ 0 \end{bmatrix}, U_b = \begin{bmatrix} \gamma & 0 & 0 \end{bmatrix}, U_d = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

Preferences: $\Lambda = 0, \Pi = 1, \Delta_h = 0, \Theta_h = 0$.

We set parameters

$\alpha = 10, \beta = 0.95, \rho_1 = 0.9, \rho_2 = 0, \sigma = 1$

(The value of γ does not affect the optimal decision rule)

The chosen matrices mean that the household's technology is:

$$c_t + k_{t-1} = i_t + y_t$$

$$\frac{k_t}{R} = i_t$$

$$l_t^2 = (0.00001)^2 i_t$$

Combining the first two of these gives the budget constraint of the permanent income model, where $k_t = b_{t+1}$.

The third equation is a very small penalty on debt-accumulation to rule out Ponzi schemes.

We set up this instance of the DLE class below:

```
α, β, ρ_1, ρ_2, σ = 10, 0.95, 0.9, 0, 1

γ = np.array([[-1], [0]])
ϕ_c = np.array([[1], [0]])
ϕ_g = np.array([[0], [1]])
ϕ_1 = 1e-5
ϕ_i = np.array([[-1], [-ϕ_1]])
δ_k = np.array([[0]])
θ_k = np.array([[1 / β]])
β = np.array([[β]])
l_λ = np.array([[0]])
π_h = np.array([[1]])
δ_h = np.array([[0]])
θ_h = np.array([[0]])

a22 = np.array([[1,   0,   0],
                [α, ρ_1, ρ_2],
                [0, 1, 0]])

c2 = np.array([[0], [σ], [0]])
ud = np.array([[0, 1, 0],
               [0, 0, 0]])
ub = np.array([[100, 0, 0]])

x0 = np.array([[0], [0], [1], [0], [0]])

info1 = (a22, c2, ub, ud)
tech1 = (ϕ_c, ϕ_g, ϕ_i, γ, δ_k, θ_k)
pref1 = (β, l_λ, π_h, δ_h, θ_h)
econ1 = DLE(info1, tech1, pref1)
```

To check the solution of this model with that from the **LQ** problem, we select the S_c matrix from the DLE class.

The solution to the DLE economy has:

$$c_t = S_c x_t$$

```
econ1.Sc
```

```
array([[ 0.    , -0.05  , 65.5172,  0.3448,  0.    ]])
```

The state vector in the DLE class is:

$$x_t = \begin{bmatrix} h_{t-1} \\ k_{t-1} \\ z_t \end{bmatrix}$$

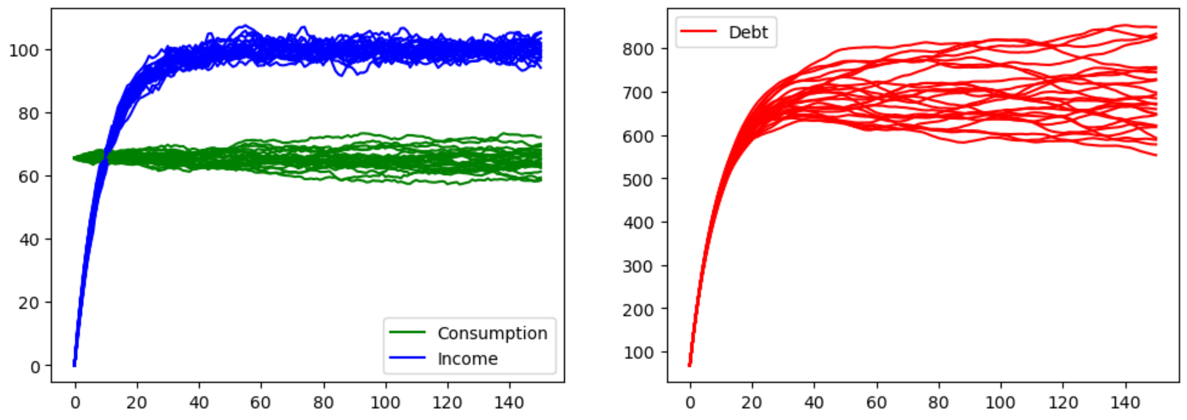
where $k_{t-1} = b_t$ is set up to be b_t in the permanent income model.

The state vector in the LQ problem is $\begin{bmatrix} z_t \\ b_t \end{bmatrix}$.

Consequently, the relevant elements of `econ1.Sc` are the same as in $-F$ occur when we apply other approaches to the same model in the lecture Optimal Savings II: LQ Techniques and this Jupyter notebook.

The plot below quickly replicates the first two figures of that lecture and that notebook to confirm that the solutions are the same

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

for i in range(25):
    econ1.compute_sequence(x0, ts_length=150)
    ax1.plot(econ1.c[0], c='g')
    ax1.plot(econ1.d[0], c='b')
ax1.plot(econ1.c[0], label='Consumption', c='g')
ax1.plot(econ1.d[0], label='Income', c='b')
ax1.legend()

for i in range(25):
    econ1.compute_sequence(x0, ts_length=150)
    ax2.plot(econ1.k[0], color='r')
ax2.plot(econ1.k[0], label='Debt', c='r')
ax2.legend()
plt.show()
```



CHAPTER

TWENTYFOUR

ROSEN SCHOOLING MODEL

Contents

This lecture is yet another part of a suite of lectures that use the quantecon DLE class to instantiate models within the [Hansen and Sargent, 2013] class of models described in detail in *Recursive Models of Dynamic Linear Economies*.

In addition to what's included in Anaconda, this lecture uses the quantecon library

```
!pip install --upgrade quantecon
```

We'll also need the following imports:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import namedtuple
from quantecon import DLE
```

24.1 A One-Occupation Model

Ryoo and Rosen's (2004) [Ryoo and Rosen, 2004] partial equilibrium model determines

- a stock of "Engineers" N_t
- a number of new entrants in engineering school, n_t
- the wage rate of engineers, w_t

It takes k periods of schooling to become an engineer.

The model consists of the following equations:

- a demand curve for engineers:

$$w_t = -\alpha_d N_t + \epsilon_{dt}$$

- a time-to-build structure of the education process:

$$N_{t+k} = \delta_N N_{t+k-1} + n_t$$

- a definition of the discounted present value of each new engineering student:

$$v_t = \beta_k \mathbb{E} \sum_{j=0}^{\infty} (\beta \delta_N)^j w_{t+k+j}$$

- a supply curve of new students driven by present value v_t:

$$n_t = \alpha_s v_t + \epsilon_{st}$$

24.2 Mapping into HS2013 Framework

We represent this model in the [Hansen and Sargent, 2013] framework by

- sweeping the time-to-build structure and the demand for engineers into the household technology, and
- putting the supply of engineers into the technology for producing goods

24.2.1 Preferences

$$\Pi = 0, \Lambda = \begin{bmatrix} \alpha_d & 0 & \cdots & 0 \end{bmatrix}, \Delta_h = \begin{bmatrix} \delta_N & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}, \Theta_h = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

where Λ is a k+1 x 1 matrix, Δ_h is a k_1 x k+1 matrix, and Θ_h is a k+1 x 1 matrix.

This specification sets $N_t = h_{1t-1}$, $n_t = c_t$, $h_{\tau+1,t-1} = n_{t-(k-\tau)}$ for $\tau = 1, ..., k$.

Below we set things up so that the number of years of education, k, can be varied.

24.2.2 Technology

To capture Ryoo and Rosen's [Ryoo and Rosen, 2004] supply curve, we use the physical technology:

$$c_t = i_t + d_{1t}$$

$$\psi_1 i_t = g_t$$

where ψ_1 is inversely proportional to α_s.

24.2.3 Information

Because we want $b_t = \epsilon_{dt}$ and $d_{1t} = \epsilon_{st}$, we set

$$A_{22} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \rho_s & 0 \\ 0 & 0 & \rho_d \end{bmatrix}, C_2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, U_b = \begin{bmatrix} 30 & 0 & 1 \end{bmatrix}, U_d = \begin{bmatrix} 10 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where ρ_s and ρ_d describe the persistence of the supply and demand shocks

```
Information = namedtuple('Information', ['a22', 'c2','ub','ud'])
Technology = namedtuple('Technology', ['ϕ_c', 'ϕ_g', 'ϕ_i', 'γ', 'δ_k', 'θ_k'])
Preferences = namedtuple('Preferences', ['β', 'l_λ', 'π_h', 'δ_h', 'θ_h'])
```

24.2.4 Effects of Changes in Education Technology and Demand

We now study how changing

- the number of years of education required to become an engineer and
- the slope of the demand curve

affects responses to demand shocks.

To begin, we set $k = 4$ and $\alpha_d = 0.1$

```
k = 4  # Number of periods of schooling required to become an engineer

β = np.array([[1 / 1.05]])
α_d = np.array([[0.1]])
α_s = 1
ε_1 = 1e-7
λ_1 = np.full((1, k), ε_1)
# Use of ε_1 is trick to aquire detectability, see HS2013 p. 228 footnote 4
l_λ = np.hstack((α_d, λ_1))
π_h = np.array([[0]])

δ_n = np.array([[0.95]])
d1 = np.vstack((δ_n, np.zeros((k - 1, 1))))
d2 = np.hstack((d1, np.eye(k)))
δ_h = np.vstack((d2, np.zeros((1, k + 1))))

θ_h = np.vstack((np.zeros((k, 1)),
                 np.ones((1, 1))))

ψ_1 = 1 / α_s

ϕ_c = np.array([[1], [0]])
ϕ_g = np.array([[0], [-1]])
ϕ_i = np.array([[-1], [ψ_1]])
γ = np.array([[0], [0]])

δ_k = np.array([[0]])
θ_k = np.array([[0]])

ρ_s = 0.8
ρ_d = 0.8

a22 = np.array([[1,   0,   0],
                [0, ρ_s,   0],
                [0,   0, ρ_d]])

c2 = np.array([[0, 0], [10, 0], [0, 10]])
ub = np.array([[30, 0, 1]])
ud = np.array([[10, 1, 0], [0, 0, 0]])

info1 = Information(a22, c2, ub, ud)
tech1 = Technology(ϕ_c, ϕ_g, ϕ_i, γ, δ_k, θ_k)
pref1 = Preferences(β, l_λ, π_h, δ_h, θ_h)

econ1 = DLE(info1, tech1, pref1)
```

We create three other instances by:

1. Raising α_d to 2
2. Raising k to 7
3. Raising k to 10

```
α_d = np.array([[2]])
l_λ = np.hstack((α_d, λ_1))
pref2 = Preferences(β, l_λ, π_h, δ_h, θ_h)
econ2 = DLE(info1, tech1, pref2)

α_d = np.array([[0.1]])

k = 7
λ_1 = np.full((1, k), ε_1)
l_λ = np.hstack((α_d, λ_1))
d1 = np.vstack((δ_n, np.zeros((k - 1, 1))))
d2 = np.hstack((d1, np.eye(k)))
δ_h = np.vstack((d2, np.zeros((1, k+1))))
θ_h = np.vstack((np.zeros((k, 1)),
                 np.ones((1, 1))))

Pref3 = Preferences(β, l_λ, π_h, δ_h, θ_h)
econ3 = DLE(info1, tech1, Pref3)

k = 10
λ_1 = np.full((1, k), ε_1)
l_λ = np.hstack((α_d, λ_1))
d1 = np.vstack((δ_n, np.zeros((k - 1, 1))))
d2 = np.hstack((d1, np.eye(k)))
δ_h = np.vstack((d2, np.zeros((1, k + 1))))
θ_h = np.vstack((np.zeros((k, 1)),
                 np.ones((1, 1))))

pref4 = Preferences(β, l_λ, π_h, δ_h, θ_h)
econ4 = DLE(info1, tech1, pref4)

shock_demand = np.array([[0], [1]])

econ1.irf(ts_length=25, shock=shock_demand)
econ2.irf(ts_length=25, shock=shock_demand)
econ3.irf(ts_length=25, shock=shock_demand)
econ4.irf(ts_length=25, shock=shock_demand)
```

The first figure plots the impulse response of n_t (on the left) and N_t (on the right) to a positive demand shock, for $\alpha_d = 0.1$ and $\alpha_d = 2$.

When $\alpha_d = 2$, the number of new students n_t rises initially, but the response then turns negative.

A positive demand shock raises wages, drawing new students into the profession.

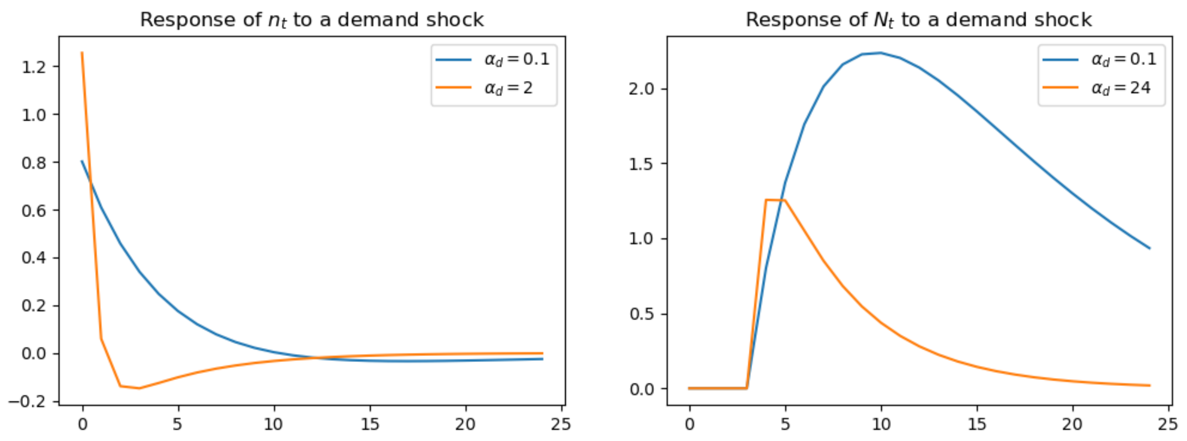
However, these new students raise N_t.

The higher is α_d, the larger the effect of this rise in N_t on wages.

This counteracts the demand shock's positive effect on wages, reducing the number of new students in subsequent periods.

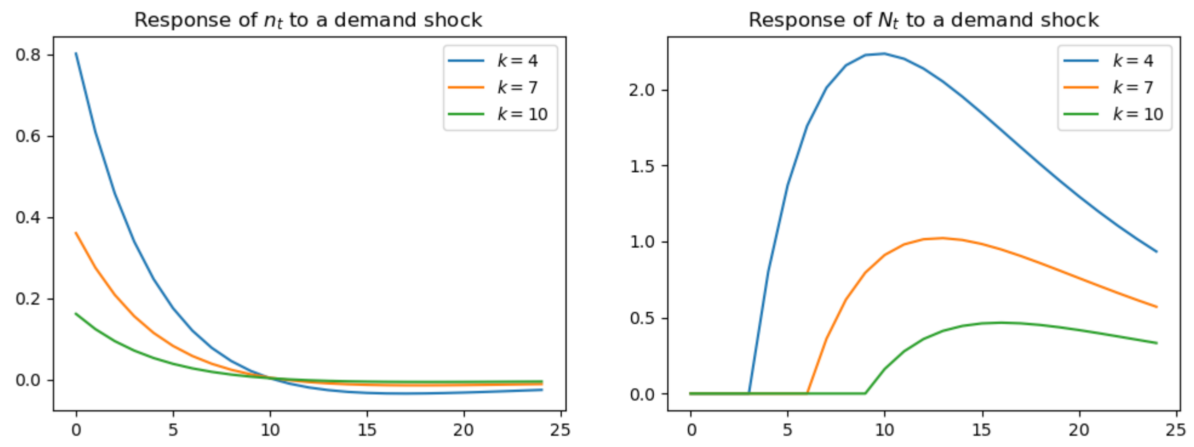
Consequently, when α_d is lower, the effect of a demand shock on N_t is larger

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(econ1.c_irf,label='$\\alpha_d = 0.1$')
ax1.plot(econ2.c_irf,label='$\\alpha_d = 2$')
ax1.legend()
ax1.set_title('Response of $n_t$ to a demand shock')

ax2.plot(econ1.h_irf[:, 0], label='$\\alpha_d = 0.1$')
ax2.plot(econ2.h_irf[:, 0], label='$\\alpha_d = 24$')
ax2.legend()
ax2.set_title('Response of $N_t$ to a demand shock')
plt.show()
```



The next figure plots the impulse response of n_t (on the left) and N_t (on the right) to a positive demand shock, for $k = 4$, $k = 7$ and $k = 10$ (with $\alpha_d = 0.1$)

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(econ1.c_irf, label='$k=4$')
ax1.plot(econ3.c_irf, label='$k=7$')
ax1.plot(econ4.c_irf, label='$k=10$')
ax1.legend()
ax1.set_title('Response of $n_t$ to a demand shock')

ax2.plot(econ1.h_irf[:,0], label='$k=4$')
ax2.plot(econ3.h_irf[:,0], label='$k=7$')
ax2.plot(econ4.h_irf[:,0], label='$k=10$')
ax2.legend()
ax2.set_title('Response of $N_t$ to a demand shock')
plt.show()
```



Both panels in the above figure show that raising k lowers the effect of a positive demand shock on entry into the engineering profession.

Increasing the number of periods of schooling lowers the number of new students in response to a demand shock.

This occurs because with longer required schooling, new students ultimately benefit less from the impact of that shock on wages.

CHAPTER

TWENTYFIVE

CATTLE CYCLES

Contents

This is another member of a suite of lectures that use the quantecon DLE class to instantiate models within the [Hansen and Sargent, 2013] class of models described in detail in *Recursive Models of Dynamic Linear Economies*.

In addition to what's in Anaconda, this lecture uses the quantecon library.

```
!pip install --upgrade quantecon
```

This lecture uses the DLE class to construct instances of the "Cattle Cycles" model of Rosen, Murphy and Scheinkman (1994) [Rosen *et al.*, 1994].

That paper constructs a rational expectations equilibrium model to understand sources of recurrent cycles in US cattle stocks and prices.

We make the following imports:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import namedtuple
from quantecon import DLE
from math import sqrt
```

25.1 The Model

The model features a static linear demand curve and a "time-to-grow" structure for cattle.

Let p_t be the price of slaughtered beef, m_t the cost of preparing an animal for slaughter, h_t the holding cost for a mature animal, $\gamma_1 h_t$ the holding cost for a yearling, and $\gamma_0 h_t$ the holding cost for a calf.

The cost processes $\{h_t, m_t\}_{t=0}^{\infty}$ are exogenous, while the price process $\{p_t\}_{t=0}^{\infty}$ is determined within a rational expectations equilibrium.

Let x_t be the breeding stock, and y_t be the total stock of cattle.

The law of motion for the breeding stock is

$$x_t = (1-\delta)x_{t-1} + gx_{t-3} - c_t$$

where $g < 1$ is the number of calves that each member of the breeding stock has each year, and c_t is the number of cattle slaughtered.

The total headcount of cattle is

$$y_t = x_t + gx_{t-1} + gx_{t-2}$$

This equation states that the total number of cattle equals the sum of adults, calves and yearlings, respectively.

A representative farmer chooses $\{c_t, x_t\}$ to maximize:

$$\mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \{p_t c_t - h_t x_t - \gamma_0 h_t (gx_{t-1}) - \gamma_1 h_t (gx_{t-2}) - m_t c_t - \frac{\psi_1}{2} x_t^2 - \frac{\psi_2}{2} x_{t-1}^2 - \frac{\psi_3}{2} x_{t-3}^2 - \frac{\psi_4}{2} c_t^2\}$$

subject to the law of motion for x_t, taking as given the stochastic laws of motion for the exogenous processes, the equilibrium price process, and the initial state $[x_{-1}, x_{-2}, x_{-3}]$.

Remark The ψ_j parameters are very small quadratic costs that are included for technical reasons to make well posed and well behaved the linear quadratic dynamic programming problem solved by the fictitious planner who in effect chooses equilibrium quantities and shadow prices.

Demand for beef is government by $c_t = a_0 - a_1 p_t + \tilde{d}_t$ where $\tilde{d}_t$ is a stochastic process with mean zero, representing a demand shifter.

25.2 Mapping into HS2013 Framework

25.2.1 Preferences

We set $\Lambda = 0, \Delta_h = 0, \Theta_h = 0, \Pi = \alpha_1^{-\frac{1}{2}}$ and $b_t = \Pi \tilde{d}_t + \Pi \alpha_0$.

With these settings, the FOC for the household's problem becomes the demand curve of the "Cattle Cycles" model.

25.2.2 Technology

To capture the law of motion for cattle, we set

$$\Delta_k = \begin{bmatrix} (1-\delta) & 0 & g \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \Theta_k = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

(where $i_t = -c_t$).

To capture the production of cattle, we set

$$\Phi_c = \begin{bmatrix} 1 \\ f_1 \\ 0 \\ 0 \\ -f_7 \end{bmatrix}, \Phi_g = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \Phi_i = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \Gamma = \begin{bmatrix} 0 & 0 & 0 \\ f_1(1-\delta) & 0 & gf_1 \\ f_3 & 0 & 0 \\ 0 & f_5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

25.2.3 Information

We set

$$A_{22} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \rho_1 & 0 & 0 \\ 0 & 0 & \rho_2 & 0 \\ 0 & 0 & 0 & \rho_3 \end{bmatrix}, C_2 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 15 \end{bmatrix}, U_b = \begin{bmatrix} \Pi\alpha_0 & 0 & 0 & \Pi \end{bmatrix}, U_d = \begin{bmatrix} 0 \\ f_2 U_h \\ f_4 U_h \\ f_6 U_h \\ f_8 U_h \end{bmatrix}$$

To map this into our class, we set $f_1^2 = \frac{\Psi_1}{2}$, $f_2^2 = \frac{\Psi_2}{2}$, $f_3^2 = \frac{\Psi_3}{2}$, $2f_1 f_2 = 1$, $2f_3 f_4 = \gamma_0 g$, $2f_5 f_6 = \gamma_1 g$.

```
# We define namedtuples in this way as it allows us to check, for example,
# what matrices are associated with a particular technology.

Information = namedtuple('Information', ['a22', 'c2', 'ub', 'ud'])
Technology = namedtuple('Technology', ['ϕ_c', 'ϕ_g', 'ϕ_i', 'γ', 'δ_k', 'θ_k'])
Preferences = namedtuple('Preferences', ['β', 'l_λ', 'π_h', 'δ_h', 'θ_h'])
```

We set parameters to those used by [Rosen *et al.*, 1994]

```
β = np.array([[0.909]])
lλ = np.array([[0]])

a1 = 0.5
πh = np.array([[1 / (sqrt(a1))]])
δh = np.array([[0]])
θh = np.array([[0]])

δ = 0.1
g = 0.85
f1 = 0.001
f3 = 0.001
f5 = 0.001
f7 = 0.001

ϕc = np.array([[1], [f1], [0], [0], [-f7]])

ϕg = np.array([[0, 0, 0, 0],
               [1, 0, 0, 0],
               [0, 1, 0, 0],
               [0, 0, 1,0],
               [0, 0, 0, 1]])

ϕi = np.array([[1], [0], [0], [0], [0]])

γ = np.array([[            0,  0,       0],
              [f1 * (1 - δ),  0, g * f1],
              [           f3,  0,       0],
              [            0, f5,       0],
              [            0,  0,       0]])

δk = np.array([[1 - δ, 0, g],
               [    1, 0, 0],
               [    0, 1, 0]])

θk = np.array([[1], [0], [0]])
```

(continues on next page)

(continued from previous page)

```
ρ1 = 0
ρ2 = 0
ρ3 = 0.6
a0 = 500
γ0 = 0.4
γ1 = 0.7
f2 = 1 / (2 * f1)
f4 = γ0 * g / (2 * f3)
f6 = γ1 * g / (2 * f5)
f8 = 1 / (2 * f7)

a22 = np.array([[1, 0, 0, 0],
                [0, ρ1, 0, 0],
                [0, 0, ρ2, 0],
                [0, 0, 0, ρ3]])

c2 = np.array([[0, 0,  0],
               [1, 0,  0],
               [0, 1,  0],
               [0, 0, 15]])

πh_scalar = πh.item()
ub = np.array([[πh_scalar * a0, 0, 0, πh_scalar]])
uh = np.array([[50, 1, 0, 0]])
um = np.array([[100, 0, 1, 0]])
ud = np.vstack(([0, 0, 0, 0],
                f2 * uh, f4 * uh, f6 * uh, f8 * um))
```

Notice that we have set $\rho_1 = \rho_2 = 0$, so h_t and m_t consist of a constant and a white noise component.

We set up the economy using tuples for information, technology and preference matrices below.

We also construct two extra information matrices, corresponding to cases when $\rho_3 = 1$ and $\rho_3 = 0$ (as opposed to the baseline case of $\rho_3 = 0.6$).

```
info1 = Information(a22, c2, ub, ud)
tech1 = Technology(ϕc, ϕg, ϕi, γ, δk, θk)
pref1 = Preferences(β, lλ, πh, δh, θh)

ρ3_2 = 1
a22_2 = np.array([[1,  0,  0,    0],
                  [0, ρ1,  0,    0],
                  [0,  0, ρ2,    0],
                  [0,  0,  0, ρ3_2]])

info2 = Information(a22_2, c2, ub, ud)

ρ3_3 = 0
a22_3 = np.array([[1,  0,  0,    0],
                  [0, ρ1,  0,    0],
                  [0,  0, ρ2,    0],
                  [0,  0,  0, ρ3_3]])

info3 = Information(a22_3, c2, ub, ud)
```

(continues on next page)

(continued from previous page)

```
# Example of how we can look at the matrices associated with a given namedtuple
info1.a22
```

```
array([[1. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 0.6]])
```

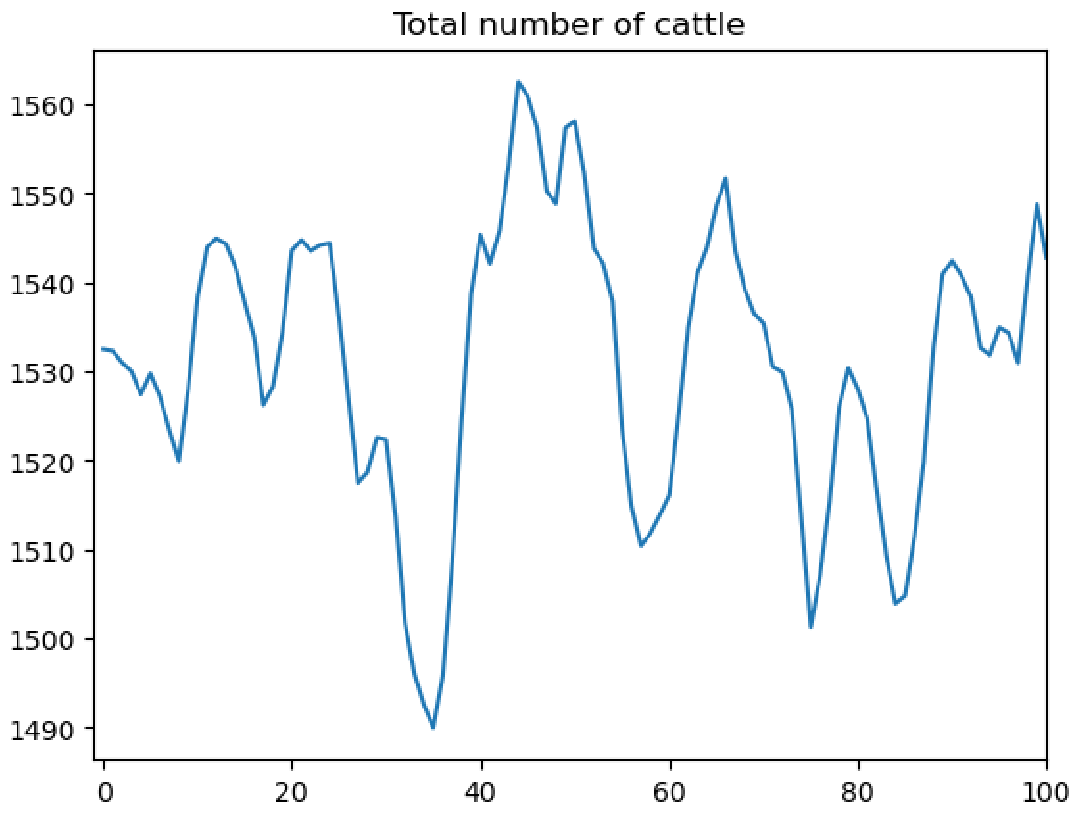
```
# Use tuples to define DLE class
econ1 = DLE(info1, tech1, pref1)
econ2 = DLE(info2, tech1, pref1)
econ3 = DLE(info3, tech1, pref1)

# Calculate steady-state in baseline case and use to set the initial condition
econ1.compute_steadystate(nnc=4)
x0 = econ1.zz
```

```
econ1.compute_sequence(x0, ts_length=100)
```

[Rosen *et al.*, 1994] use the model to understand the sources of recurrent cycles in total cattle stocks.

Plotting y_t for a simulation of their model shows its ability to generate cycles in quantities

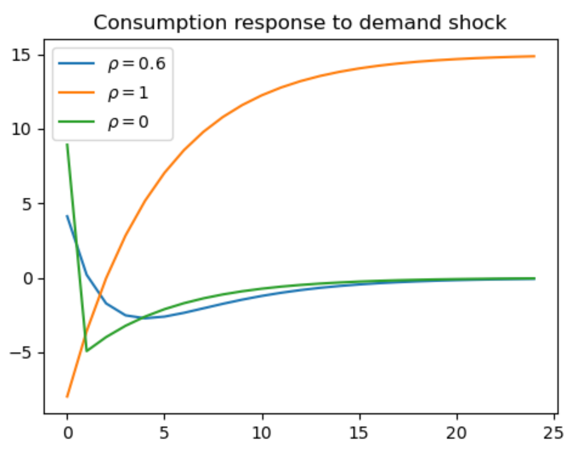
```
# Calculation of y_t
totalstock = econ1.k[0] + g * econ1.k[1] + g * econ1.k[2]
fig, ax = plt.subplots()
ax.plot(totalstock)
ax.set_xlim((-1, 100))
ax.set_title('Total number of cattle')
plt.show()
```

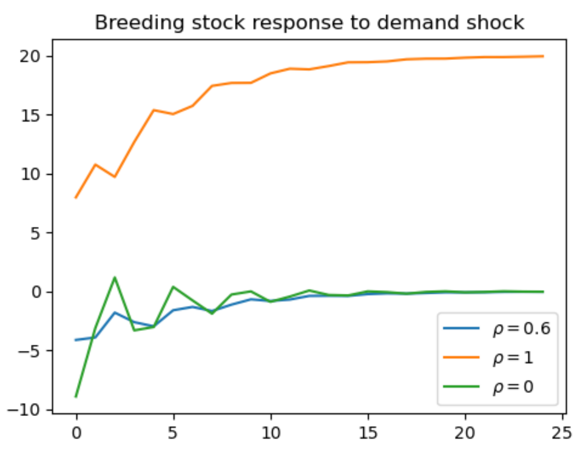


In their Figure 3, [Rosen *et al.*, 1994] plot the impulse response functions of consumption and the breeding stock of cattle to the demand shock, $\tilde{d}_t$, under the three different values of ρ_3.

We replicate their Figure 3 below

```
shock_demand = np.array([[0], [0], [1]])

econ1.irf(ts_length=25, shock=shock_demand)
econ2.irf(ts_length=25, shock=shock_demand)
econ3.irf(ts_length=25, shock=shock_demand)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(econ1.c_irf, label=r'$\rho=0.6$')
ax1.plot(econ2.c_irf, label=r'$\rho=1$')
ax1.plot(econ3.c_irf, label=r'$\rho=0$')
ax1.set_title('Consumption response to demand shock')
ax1.legend()

ax2.plot(econ1.k_irf[:, 0], label=r'$\rho=0.6$')
ax2.plot(econ2.k_irf[:, 0], label=r'$\rho=1$')
ax2.plot(econ3.k_irf[:, 0], label=r'$\rho=0$')
ax2.set_title('Breeding stock response to demand shock')
ax2.legend()
plt.show()
```





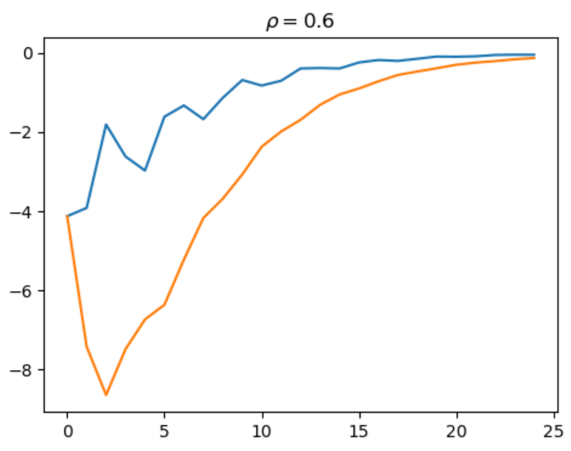
The above figures show how consumption patterns differ markedly, depending on the persistence of the demand shock:

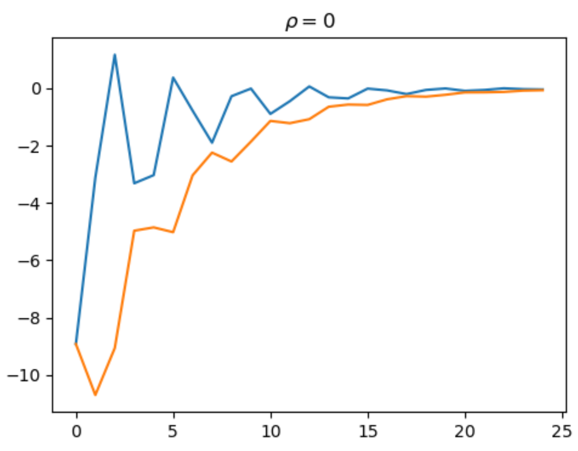
- If it is purely transitory ($\rho_3 = 0$) then consumption rises immediately but is later reduced to build stocks up again.
- If it is permanent ($\rho_3 = 1$), then consumption falls immediately, in order to build up stocks to satisfy the permanent rise in future demand.

In Figure 4 of their paper, [Rosen *et al.*, 1994] plot the response to a demand shock of the breeding stock *and* the total stock, for $\rho_3 = 0$ and $\rho_3 = 0.6$.

We replicate their Figure 4 below

```
total1_irf = econ1.k_irf[:, 0] + g * econ1.k_irf[:, 1] + g * econ1.k_irf[:, 2]
total3_irf = econ3.k_irf[:, 0] + g * econ3.k_irf[:, 1] + g * econ3.k_irf[:, 2]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(econ1.k_irf[:, 0], label='Breeding Stock')
ax1.plot(total1_irf, label='Total Stock')
ax1.set_title(r'$\rho=0.6$')

ax2.plot(econ3.k_irf[:, 0], label='Breeding Stock')
ax2.plot(total3_irf, label='Total Stock')
ax2.set_title(r'$\rho=0$')
plt.show()
```





The fact that y_t is a weighted moving average of x_t creates a humped shape response of the total stock in response to demand shocks, contributing to the cyclicality seen in the first graph of this lecture.

CHAPTER

TWENTYSIX

SHOCK NON INVERTIBILITY

Contents

26.1 Overview

This is another member of a suite of lectures that use the quantecon DLE class to instantiate models within the [Hansen and Sargent, 2013] class of models described in *Recursive Models of Dynamic Linear Economies*.

In addition to what's in Anaconda, this lecture uses the quantecon library.

```
!pip install --upgrade quantecon
```

We'll make these imports:

```
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt
from quantecon import DLE
from math import sqrt
```

This lecture describes an early contribution to what is now often called a **news and noise** issue.

In particular, it analyzes a **shock-invertibility** issue that is endemic within a class of permanent income models.

Technically, the invertibility problem indicates a situation in which histories of the shocks in an econometrician's autoregressive or Wold moving average representation span a smaller information space than do the shocks that are seen by the agents inside the econometrician's model.

An econometrician who is unaware of the problem would misinterpret shocks and likely responses to them.

A shock-invertibility that is technically close to the one studied here is discussed by Eric Leeper, Todd Walker, and Susan Yang [Leeper *et al.*, 2013] in their analysis of **fiscal foresight**.

A distinct shock-invertibility issue is present in the special LQ consumption smoothing model in this quantecon lecture *Information and Consumption Smoothing*.

26.2 Model

We consider the following modification of Robert Hall's (1978) model [Hall, 1978] in which the endowment process is the sum of two orthogonal autoregressive processes:

Preferences

$$-\frac{1}{2}\mathbb{E}\sum_{t=0}^{\infty}\beta^t[(c_t-b_t)^2+l_t^2]|J_0$$

$$s_t = c_t$$

$$b_t = U_b z_t$$

Technology

$$c_t + i_t = \gamma k_{t-1} + d_t$$

$$k_t = \delta_k k_{t-1} + i_t$$

$$g_t = \phi_1 i_t \,, \phi_1 > 0$$

$$g_t \cdot g_t = l_t^2$$

Information

$$z_{t+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} z_t + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 4 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} w_{t+1}$$

$$U_b = \begin{bmatrix} 30 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$U_d = \begin{bmatrix} 5 & 1 & 1 & 0.8 & 0.6 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The preference shock is constant at 30, while the endowment process is the sum of a constant and two orthogonal processes.

Specifically:

$$d_t = 5 + d_{1t} + d_{2t}$$

$$d_{1t} = 0.9 d_{1t-1} + w_{1t}$$

$$d_{2t} = 4w_{2t} + 0.8(4w_{2t-1}) + 0.6(4w_{2t-2}) + 0.4(4w_{2t-3})$$

d_{1t} is a first-order AR process, while d_{2t} is a third-order pure moving average process.

```
γ_1 = 0.05
γ = np.array([[γ_1], [0]])
ϕ_c = np.array([[1], [0]])
ϕ_g = np.array([[0], [1]])
ϕ_1 = 0.00001
ϕ_i = np.array([[1], [-ϕ_1]])
δ_k = np.array([[1]])
θ_k = np.array([[1]])
```

(continues on next page)

(continued from previous page)

```
β = np.array([[1 / 1.05]])
l_λ = np.array([[0]])
π_h = np.array([[1]])
δ_h = np.array([[.9]])
θ_h = np.array([[1]]) - δ_h
ud = np.array([[5, 1, 1, 0.8, 0.6, 0.4],
               [0, 0, 0,   0,   0,   0]])
a22 = np.zeros((6, 6))
# Chase's great trick
a22[[0, 1, 3, 4, 5], [0, 1, 2, 3, 4]] = np.array([1.0, 0.9, 1.0, 1.0, 1.0])
c2 = np.zeros((6, 2))
c2[[1, 2], [0, 1]] = np.array([1.0, 4.0])
ub = np.array([[30, 0, 0, 0, 0, 0]])
x0 = np.array([[5], [150], [1], [0], [0], [0], [0], [0]])

info1 = (a22, c2, ub, ud)
tech1 = (ϕ_c, ϕ_g, ϕ_i, γ, δ_k, θ_k)
pref1 = (β, l_λ, π_h, δ_h, θ_h)

econ1 = DLE(info1, tech1, pref1)
```

We define the household's net of interest deficit as $c_t - d_t$.

Hall's model imposes "expected present-value budget balance" in the sense that

$$\mathbb{E}\sum_{j=0}^{\infty}\beta^j(c_{t+j} - d_{t+j})|J_t = \beta^{-1}k_{t-1}\ \forall t$$

Define a moving average representation of $(c_t, c_t - d_t)$ in terms of the w_ts to be:

$$\begin{bmatrix} c_t \\ c_t - d_t \end{bmatrix} = \begin{bmatrix} \sigma_1(L) \\ \sigma_2(L) \end{bmatrix} w_t$$

Hall's model imposes the restriction $\sigma_2(\beta) = [0\ \ 0]$.

- The consumer who lives inside this model observes histories of both components of the endowment process d_{1t} and d_{2t}.
- The econometrician has data on the history of the pair $[c_t, d_t]$, but not directly on the history of w_t's.
- The econometrician obtains a Wold representation for the process $[c_t, c_t - d_t]$:

$$\begin{bmatrix} c_t \\ c_t - d_t \end{bmatrix} = \begin{bmatrix} \sigma_1^*(L) \\ \sigma_2^*(L) \end{bmatrix} u_t$$

A representation with equivalent shocks would be recovered by estimating a bivariate vector autoregression for $c_t, c_t - d_t$.

The Appendix of chapter 8 of [Hansen and Sargent, 2013] explains why the impulse response functions in the Wold representation estimated by the econometrician do not resemble the impulse response functions that depict the response of consumption and the net-of-interest deficit to innovations w_t to the consumer's information.

Technically, $\sigma_2(\beta) = [0\ \ 0]$ implies that the history of u_ts spans a *smaller* linear space than does the history of w_ts.

This means that u_t will typically be a distributed lag of w_t that is not concentrated at zero lag:

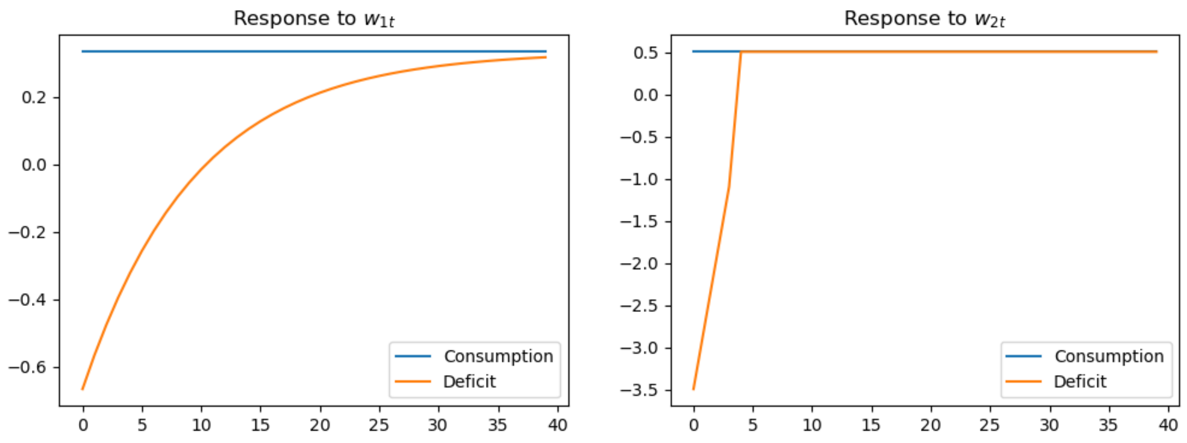
$$u_t = \sum_{j=0}^{\infty}\alpha_j w_{t-j}$$

Thus, the econometrician's news u_t typically responds belatedly to the consumer's news w_t.

26.3 Code

We will construct Figures from Chapter 8 Appendix E of [Hansen and Sargent, 2013] to illustrate these ideas:

```python
# This is Fig 8.E.1 from p.188 of HS2013

econ1.irf(ts_length=40, shock=None)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(econ1.c_irf, label='Consumption')
ax1.plot(econ1.c_irf - econ1.d_irf[:,0].reshape(40,1), label='Deficit')
ax1.legend()
ax1.set_title('Response to $w_{1t}$')

shock2 = np.array([[0], [1]])
econ1.irf(ts_length=40, shock=shock2)

ax2.plot(econ1.c_irf, label='Consumption')
ax2.plot(econ1.c_irf - econ1.d_irf[:,0].reshape(40, 1), label='Deficit')
ax2.legend()
ax2.set_title('Response to $w_{2t}$')
plt.show()
```



The above figure displays the impulse response of consumption and the net-of-interest deficit to the innovations w_t to the consumer's non-financial income or endowment process.

Consumption displays the characteristic "random walk" response with respect to each innovation.

Each endowment innovation leads to a temporary surplus followed by a permanent net-of-interest deficit.

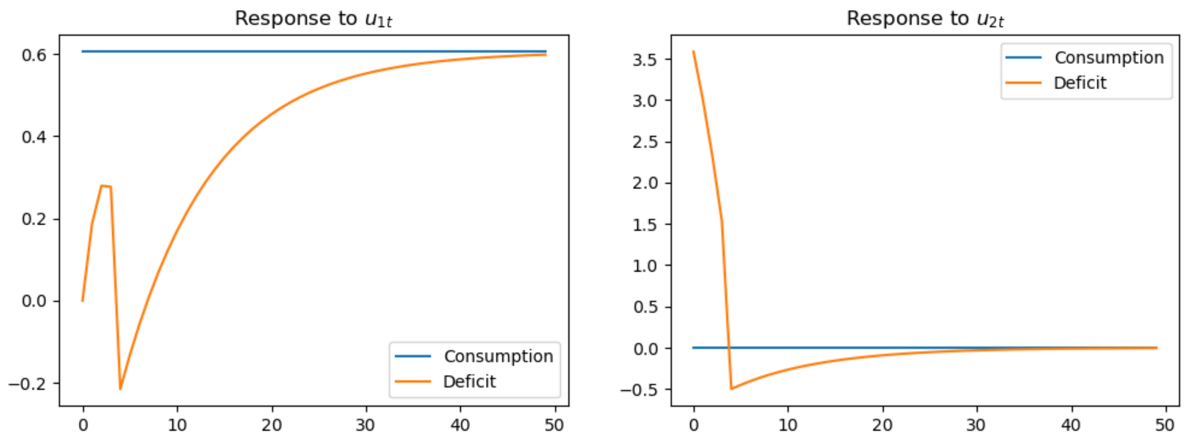
The temporary surplus just offsets the permanent deficit in terms of expected present value.

```python
G_HS = np.vstack([econ1.Sc, econ1.Sc-econ1.Sd[0, :].reshape(1, 8)])
H_HS = 1e-8 * np.eye(2)  # Set very small so there is no measurement error
lss_hs = qe.LinearStateSpace(econ1.A0, econ1.C, G_HS, H_HS)

hs_kal = qe.Kalman(lss_hs)
w_lss = hs_kal.whitener_lss()
ma_coefs = hs_kal.stationary_coefficients(50, 'ma')

# This is Fig 8.E.2 from p.189 of HS2013
```

(continues on next page)

(continued from previous page)

```
ma_coefs = ma_coefs
jj = 50
y1_w1 = np.empty(jj)
y2_w1 = np.empty(jj)
y1_w2 = np.empty(jj)
y2_w2 = np.empty(jj)

for t in range(jj):
    y1_w1[t] = ma_coefs[t][0, 0]
    y1_w2[t] = ma_coefs[t][0, 1]
    y2_w1[t] = ma_coefs[t][1, 0]
    y2_w2[t] = ma_coefs[t][1, 1]

# This scales the impulse responses to match those in the book
y1_w1 = sqrt(hs_kal.stationary_innovation_covar()[0, 0]) * y1_w1
y2_w1 = sqrt(hs_kal.stationary_innovation_covar()[0, 0]) * y2_w1
y1_w2 = sqrt(hs_kal.stationary_innovation_covar()[1, 1]) * y1_w2
y2_w2 = sqrt(hs_kal.stationary_innovation_covar()[1, 1]) * y2_w2

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(y1_w1, label='Consumption')
ax1.plot(y2_w1, label='Deficit')
ax1.legend()
ax1.set_title('Response to $u_{1t}$')

ax2.plot(y1_w2, label='Consumption')
ax2.plot(y2_w2, label='Deficit')
ax2.legend()
ax2.set_title('Response to $u_{2t}$')
plt.show()
```



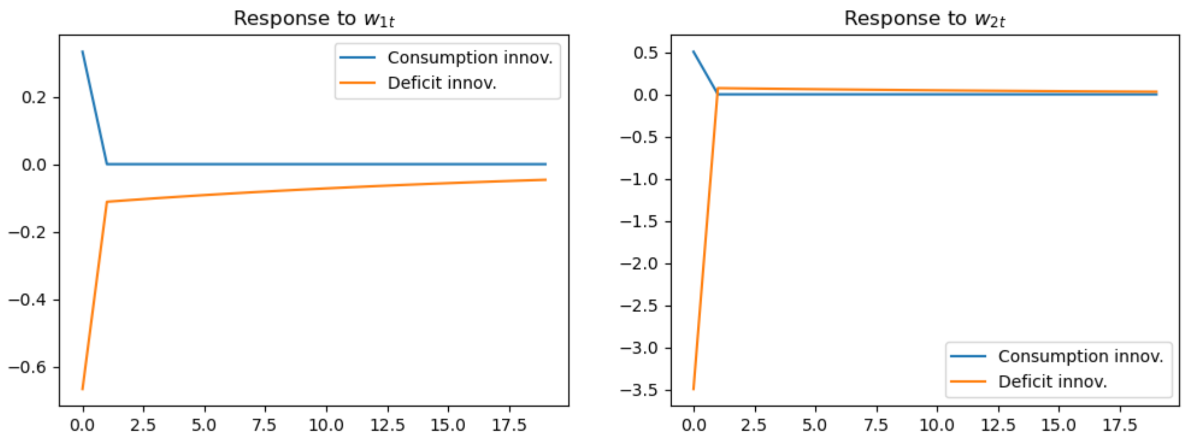
The above figure displays the impulse response of consumption and the deficit to the innovations in the econometrician's Wold representation

- this is the object that would be recovered from a high order vector autoregression on the econometrician's observations.

Consumption responds only to the first innovation

- this is indicative of the Granger causality imposed on the $[c_t, c_t - d_t]$ process by Hall's model: consumption Granger causes $c_t - d_t$, with no reverse causality.

```
# This is Fig 8.E.3 from p.189 of HS2013

jj = 20
irf_wlss = w_lss.impulse_response(jj)
ycoefs = irf_wlss[1]
# Pull out the shocks
a1_w1 = np.empty(jj)
a1_w2 = np.empty(jj)
a2_w1 = np.empty(jj)
a2_w2 = np.empty(jj)

for t in range(jj):
    a1_w1[t] = ycoefs[t][0, 0]
    a1_w2[t] = ycoefs[t][0, 1]
    a2_w1[t] = ycoefs[t][1, 0]
    a2_w2[t] = ycoefs[t][1, 1]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(a1_w1, label='Consumption innov.')
ax1.plot(a2_w1, label='Deficit innov.')
ax1.set_title('Response to $w_{1t}$')
ax1.legend()
ax2.plot(a1_w2, label='Consumption innov.')
ax2.plot(a2_w2, label='Deficit innov.')
ax2.legend()
ax2.set_title('Response to $w_{2t}$')
plt.show()
```



The above figure displays the impulse responses of u_t to w_t, as depicted in:

$$u_t = \sum_{j=0}^{\infty} \alpha_j w_{t-j}$$

While the responses of the innovations to consumption are concentrated at lag zero for both components of w_t, the responses of the innovations to $(c_t - d_t)$ are spread over time (especially in response to $w_{1t}$).

Thus, the innovations to $(c_t - d_t)$ as revealed by the vector autoregression depend on what the economic agent views as "old news".

Part IV

Other

CHAPTER

TWENTYSEVEN

TROUBLESHOOTING

Contents

This page is for readers experiencing errors when running the code from the lectures.

27.1 Fixing Your Local Environment

The basic assumption of the lectures is that code in a lecture should execute whenever

1. it is executed in a Jupyter notebook and
2. the notebook is running on a machine with the latest version of Anaconda Python.

You have installed Anaconda, haven't you, following the instructions in this lecture?

Assuming that you have, the most common source of problems for our readers is that their Anaconda distribution is not up to date.

Here's a useful article on how to update Anaconda.

Another option is to simply remove Anaconda and reinstall.

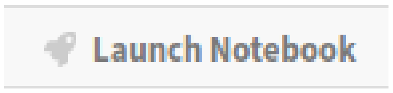
You also need to keep the external code libraries, such as QuantEcon.py up to date.

For this task you can either

- use conda install -y quantecon on the command line, or
- execute !conda install -y quantecon within a Jupyter notebook.

If your local environment is still not working you can do two things.

First, you can use a remote machine instead, by clicking on the Launch Notebook icon available for each lecture



Second, you can report an issue, so we can try to fix your local set up.

We like getting feedback on the lectures so please don't hesitate to get in touch.

27.2 Reporting an Issue

One way to give feedback is to raise an issue through our issue tracker.

Please be as specific as possible. Tell us where the problem is and as much detail about your local set up as you can provide.

Another feedback option is to use our discourse forum.

Finally, you can provide direct feedback to contact@quantecon.org

CHAPTER

TWENTYEIGHT

REFERENCES

CHAPTER

TWENTYNINE

EXECUTION STATISTICS

This table contains the latest execution statistics.

Document	Modified	Method	Run Time (s)	Status
cattle_cycles	2024-05-01 02:00	cache	10.04	✅
cons_news	2024-05-01 02:00	cache	5.89	✅
cross_product_trick	2024-05-01 02:00	cache	0.85	✅
growth_in_dles	2024-05-01 02:00	cache	5.44	✅
hs_invertibility_example	2024-05-01 02:00	cache	5.59	✅
hs_recursive_models	2024-05-01 02:00	cache	0.96	✅
intro	2024-05-01 02:00	cache	0.96	✅
irfs_in_hall_model	2024-05-01 02:00	cache	5.39	✅
kalman	2024-05-01 02:00	cache	7.04	✅
kalman_2	2024-05-01 02:01	cache	22.73	✅
lagrangian_lqdp	2024-05-01 02:01	cache	19.62	✅
linear_models	2024-05-01 02:01	cache	6.78	✅
lq_inventories	2024-05-01 02:01	cache	12.37	✅
lqcontrol	2024-05-01 02:02	cache	5.68	✅
lqramsey	2024-05-01 02:02	cache	6.87	✅
lucas_asset_pricing_dles	2024-05-01 02:02	cache	5.36	✅
markov_jump_lq	2024-05-01 02:03	cache	73.45	✅
muth_kalman	2024-05-01 02:03	cache	5.55	✅
perm_income	2024-05-01 02:03	cache	3.12	✅
perm_income_cons	2024-05-01 02:03	cache	6.63	✅
permanent_income_dles	2024-05-01 02:03	cache	5.49	✅
rosen_schooling_model	2024-05-01 02:03	cache	5.25	✅
smoothing	2024-05-01 02:04	cache	5.58	✅
smoothing_tax	2024-05-01 02:04	cache	8.09	✅
status	2024-05-01 02:00	cache	0.96	✅
tax_smoothing_1	2024-05-01 02:04	cache	11.71	✅
tax_smoothing_2	2024-05-01 02:04	cache	6.07	✅
tax_smoothing_3	2024-05-01 02:04	cache	5.96	✅
troubleshooting	2024-05-01 02:00	cache	0.96	✅
zreferences	2024-05-01 02:00	cache	0.96	✅

These lectures are built on `linux` instances through `github actions` so are executed using the following hardware specifications

BIBLIOGRAPHY

[AM05] D. B. O. Anderson and J. B. Moore. *Optimal Filtering*. Dover Publications, 2005.

[AHMS96] E. W. Anderson, L. P. Hansen, E. R. McGrattan, and T. J. Sargent. Mechanics of Forming and Estimating Dynamic Linear Economies. In *Handbook of Computational Economics*. Elsevier, vol 1 edition, 1996.

[AP11] Orazio P Attanasio and Nicola Pavoni. Risk sharing in private information models with asset accumulation: explaining the excess smoothness of consumption. *Econometrica*, 79(4):1027–1068, 2011.

[Bar79] Robert J Barro. On the Determination of the Public Debt. *Journal of Political Economy*, 87(5):940–971, 1979.

[Bar99] Robert J Barro. Determinants of democracy. *Journal of Political economy*, 107(S6):S158–S183, 1999.

[BM03] Robert J Barro and Rachel McCleary. Religion and economic growth. Technical Report, National Bureau of Economic Research, 2003.

[Bew86] Truman F Bewley. Stationary monetary equilibrium with a continuum of independently fluctuating consumers. In Werner Hildenbran and Andreu Mas-Colell, editors, *Contributions to Mathematical Economics in Honor of Gerard Debreu*, pages 27–102. North-Holland, Amsterdam, 1986.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Car01] Christopher D Carroll. A Theory of the Consumption Function, with and without Liquidity Constraints. *Journal of Economic Perspectives*, 15(3):23–45, 2001.

[Dea91] Angus Deaton. Saving and Liquidity Constraints. *Econometrica*, 59(5):1221–1248, 1991.

[DP94] Angus Deaton and Christina Paxson. Intertemporal Choice and Inequality. *Journal of Political Economy*, 102(3):437–467, 1994.

[DVGC99] JBR Do Val, JC Geromel, and OLV Costa. Solutions for the linear-quadratic control problem of markov jump linear systems. *Journal of Optimization Theory and Applications*, 103(2):283–311, 1999.

[EG87] Robert F Engle and Clive W J Granger. Co-integration and Error Correction: Representation, Estimation, and Testing. *Econometrica*, 55(2):251–276, 1987.

[Fri56] M. Friedman. *A Theory of the Consumption Function*. Princeton University Press, 1956.

[Gal37] Albert Gallatin. Report on the finances**, november, 1807. In *Reports of the Secretary of the Treasury of the United States, Vol 1*. Government printing office, Washington, DC, 1837.

[Hal78] Robert E Hall. Stochastic Implications of the Life Cycle-Permanent Income Hypothesis: Theory and Evidence. *Journal of Political Economy*, 86(6):971–987, 1978.

[HM82] Robert E Hall and Frederic S Mishkin. The Sensitivity of Consumption to Transitory Income: Estimates from Panel Data on Households. *National Bureau of Economic Research Working Paper Series*, 1982.

[HS08] L P Hansen and T J Sargent. *Robustness*. Princeton University Press, 2008.

[HS13a] L P Hansen and T J Sargent. *Recursive Models of Dynamic Linear Economies*. The Gorman Lectures in Economics. Princeton University Press, 2013.

[HS13b] Lars Peter Hansen and Thomas J. Sargent. *Recursive Linear Models of Dynamic Economics*. Princeton University Press, Princeton, New Jersey, 2013.

[HLL96] O Hernandez-Lerma and J B Lasserre. *Discrete-Time Markov Control Processes: Basic Optimality Criteria*. Number Vol 1 in Applications of Mathematics Stochastic Modelling and Applied Probability. Springer, 1996.

[HMMS60] Charles Holt, Franco Modigliani, John F. Muth, and Herbert Simon. *Planning Production, Inventories, and Work Force*. Prentice-Hall International Series in Management, New Jersey, 1960.

[JYC88] Robert J. Shiller John Y. Campbell. The Dividend-Price Ratio and Expectations of Future Dividends and Discount Factors. *Review of Financial Studies*, 1(3):195–228, 1988.

[LWY13] Eric M. Leeper, Todd B. Walker, and Shu-Chun Susan Yang. Fiscal foresight and information flows. *Econometrica*, 81(3):1115–1145, May 2013.

[LL01] Martin Lettau and Sydney Ludvigson. Consumption, Aggregate Wealth, and Expected Stock Returns. *Journal of Finance*, 56(3):815–849, 06 2001.

[LL04] Martin Lettau and Sydney C. Ludvigson. Understanding Trend and Cycle in Asset Values: Reevaluating the Wealth Effect on Consumption. *American Economic Review*, 94(1):276–299, March 2004.

[LS18] L Ljungqvist and T J Sargent. *Recursive Macroeconomic Theory*. MIT Press, 4 edition, 2018.

[Luc78] Robert E Lucas, Jr. Asset prices in an exchange economy. *Econometrica: Journal of the Econometric Society*, 46(6):1429–1445, 1978.

[LS83] Robert E Lucas, Jr. and Nancy L Stokey. Optimal Fiscal and Monetary Policy in an Economy without Capital. *Journal of monetary Economics*, 12(3):55–93, 1983.

[MB54] F. Modigliani and R. Brumberg. Utility analysis and the consumption function: An interpretation of cross-section data. In K.K Kurihara, editor, *Post-Keynesian Economics*. 1954.

[Mut60] John F Muth. Optimal properties of exponentially weighted forecasts. *Journal of the american statistical association*, 55(290):299–306, 1960.

[Par99] Jonathan A Parker. The Reaction of Household Consumption to Predictable Changes in Social Security Taxes. *American Economic Review*, 89(4):959–973, 1999.

[Ram27] F. P. Ramsey. A Contribution to the theory of taxation. *Economic Journal*, 37(145):47–61, 1927.

[RMS94] Sherwin Rosen, Kevin M Murphy, and Jose A Scheinkman. Cattle cycles. *Journal of Political Economy*, 102(3):468–492, 1994.

[RR04] Jaewoo Ryoo and Sherwin Rosen. The engineering labor market. *Journal of political economy*, 112(S1):S110–S140, 2004.

[SHR91] Thomas Sargent, Lars Peter Hansen, and Will Roberts. Observable implications of present value budget balance. In *Rational Expectations Econometrics*. Westview Press, 1991.

[STY04] Kjetil Storesletten, Christopher I Telmer, and Amir Yaron. Consumption and risk sharing over the life cycle. *Journal of Monetary Economics*, 51(3):609–633, 2004.

[SW09] Lars E.O. Svensson and Noah Williams. Optimal Monetary Policy under Uncertainty in DSGE Models: A Markov Jump-Linear-Quadratic Approach. In Klaus Schmidt-Hebbel, Carl E. Walsh, Norman Loayza (Series Editor), and Klaus Schmidt-Hebbel (Series, editors, *Monetary Policy under Uncertainty and Learning*, volume 13 of Central Banking, Analysis, and Economic Policies Book Series, chapter 3, pages 077–114. Central Bank of Chile, edition, March 2009.

[SW+08] Lars EO Svensson, Noah Williams, and others. Optimal monetary policy under uncertainty: a markov jump-linear-quadratic approach. *Federal Reserve Bank of St. Louis Review*, 90(4):275–293, 2008.

INDEX