Part II

LQ Control

CHAPTER

FIVE

INFORMATION AND CONSUMPTION SMOOTHING

Contents

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

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

5.1 Overview

This lecture studies 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 in the quantecon lecture.

The two consumers have different information about future nonfinancial incomes.

One 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.

The other, less well informed, consumer each period receives a shock that equals the part of today's nonfinancial income that could not be forecast from all 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 below.

This difference in their information sets manifests itself in their responding differently to what they regard as time t information shocks.

Thus, while they receive exactly the same histories of nonfinancial income, our two consumers receive different **shocks** or **news** about their **future** nonfinancial incomes.

We compare behaviors of our two consumers as a way to learn about

- operating characteristics of the linear-quadratic permanent income model

- how the Kalman filter introduced in this lecture and/or the theory of optimal forecasting introduced in *this lecture* embody lessons that can be applied to the **news** and **noise** literature
- various ways of representing and computing optimal decision rules in the linear-quadratic permanent income model
- a **Ricardian equivalence** outcome describing 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 some of the **invertibility** issues that take center stage in the analysis of **fiscal foresight** by Eric Leeper, Todd Walker, and Susan Yang [LWY13].

5.2 Two Representations of the Same Nonfinancial Income Process

Where $\beta \in (0, 1)$, we study consequences of endowing a consumer with one of the two alternative representations for the change in the consumer's nonfinancial income $y_{t+1} - y_t$.

The first representation, which we shall refer to as the **original representation**, is

$$y_{t+1} - y_t = \epsilon_{t+1} - \beta^{-1}\epsilon_t \tag{1}$$

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 original shock ϵ_t and can use both of them to forecast future y_{t+j}'s.

Furthermore, as we'll see below, representation (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

$$y_{t+1} - y_t = a_{t+1} - \beta a_t \tag{2}$$

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 two 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}.

The second representation is the **innovations representation** from Kalman filtering theory.

To see how this works, note that equating representations (1) and (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,

$$a_{t+1} = \epsilon_{t+1} + (\beta - \beta^{-1})\sum_{j=0}^{\infty}\beta^j\epsilon_{t-j} \tag{3}$$

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 the coefficient $g_0 = \beta^{-2} \sigma_\epsilon^2$ and that $g_j = 0$ for $j \neq 0$.

Alternatively, if you are uncomfortable with covariance generating functions, note that we can directly calculate σ_a^2 from formula (3) according to

$$\sigma_a^2 = \sigma_\epsilon^2 + [1 + (\beta - \beta^{-1})^2 \sum_{j=0}^{\infty} \beta^{2j}] = \beta^{-1} \sigma_\epsilon^2.$$

5.2.1 Application of Kalman filter

We can also obtain representation (2) from representation (1) by using the **Kalman filter**.

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

Compute the steady-state Kalman filter for this system and let K be the steady-state gain and a_{t+1} the one-step ahead innovation.

The innovations representation is

$$\begin{aligned} \hat{\epsilon}_{t+1} &= 0 \hat{\epsilon}_t + K a_{t+1} \\ z_{t+1} &= -\beta a_t + a_{t+1} \end{aligned}$$

By applying formulas for the steady-state Kalman filter, by hand we computed that $K = \beta^2, \sigma_a^2 = \beta^{-2} \sigma_\epsilon^2 = \beta^{-2}$, and $\Sigma = (1 - \beta^2) \sigma_\epsilon^2$.

We can also obtain these formulas via the classical filtering theory described in *this lecture*.

5.2.2 News Shocks and Less Informative Shocks

Representation (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 having all sorts of information about the income process.

Representation (2) for the **same** income process is driven by shocks a_t that contain less information than the news shock ϵ_t.

Representation (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 contained solely 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 $y_t, y_{t-1}, \ldots$.

The shock ϵ_t is **not fundamental** and has more information about the future of the $\{y_t - y_{t-1}\}$ process than is contained in a_t.

Representation (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 $y^t = [y_t, y_{t-1}, ...]$ of current and past y's.

Staring at representation (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 **better informed** representation (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 an **increase** 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 **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 successors created by the roll-over operation.
- In all future 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 (2), i.e., the innovation 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 $j = 1, 2, \ldots$.

The present value of the impulse response or moving average coefficients for representation (2) is $d_a(\beta) = \frac{1-\beta^2}{1-\beta} = (1+\beta)$, another fact that will be important below.

5.2.3 Representation of ϵ_t in Terms of Future y's

Notice that reprentation (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 eventually 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 **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 contains **exact** information about an important linear combination of **future** nonfinancial income.

5.2.4 Representation in Terms of a_t Shocks

Next notice that representation (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

$$a_{t+1} = y_{t+1} - (1-\beta)\sum_{j=0}^{\infty}\beta^j y_{t-j}$$

and where the information set is $y^t = [y_t, y_{t-1}, ...]$, the one step-ahead optimal prediction is

$$E[y_{t+1}|y^t] = (1-\beta)\sum_{j=0}^{\infty}\beta^j y_{t-j}$$

5.2.5 Permanent Income Consumption-Smoothing Model

When we computed optimal consumption-saving policies for the two representations using formulas obtained with the difference equation approach described in the quantecon lecture, we obtain:

for a consumer having the information assumed in the news representation (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 (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 the Python programs to be reported below using state-space representations and dynamic programming.

Evidently the two consumers behave differently though they receive exactly the same histories of nonfinancial income.

The consumer with information associated with representation (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 consumer with information associated with representation (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.

We can regard the first consumer as someone whose behavior sharply illustrates the behavior assumed in a classic Ricardian equivalence experiment.

5.3 State Space Representations

We can cast our two representations in terms of the following two state space systems

$$\begin{bmatrix} y_{t+1} \\ \epsilon_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & -\beta^{-1} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ \epsilon_t \end{bmatrix} + \begin{bmatrix} \sigma_\epsilon \\ \sigma_\epsilon \end{bmatrix} v_{t+1}$$

$$y_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ \epsilon_t \end{bmatrix}$$

and

$$\begin{bmatrix} y_{t+1} \\ a_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & -\beta \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ a_t \end{bmatrix} + \begin{bmatrix} \sigma_a \\ \sigma_a \end{bmatrix} u_{t+1}$$

$$y_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_t \\ a_t \end{bmatrix}$$

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 the quantecon lecture.

All the code that we shall use below is presented in that lecture.

5.3.1 Computations

We shall use Python to form **both** of the above two state-space representations, using 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 the 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 following initial conditions $x_0 =$

$$\begin{bmatrix} 10 \\ 0 \end{bmatrix}$$

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 (later I'll give some more details about how we'll run some special versions of these)

We want to solve the LQ problem:

$$\min \sum_{t=0}^{\infty} \beta^t \left(c_t - \gamma\right)^2$$

subject to the 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
%matplotlib inline
```

```
# 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.])
```

```
# Original representation state transition matrices
ALQ1 = np.array([[1, -R, 0],
                 [0, 0, 0],
                 [-R, 0, R]])
```

(continues on next page)

(continued from previous page)

```
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 (1) are

$$\begin{aligned} c_t^* &= y_t - \epsilon_t - (1-\beta)\, b_t, \\ b_{t+1}^* &= \beta^{-1} c_t^* + \beta^{-1} b_t - \beta^{-1} y_t \\ &= \beta^{-1} y_t - \beta^{-1}\epsilon_t - \left(\beta^{-1} - 1\right) b_t + \beta^{-1} b_t - \beta^{-1} y_t \\ &= b_t - \beta^{-1}\epsilon_t. \end{aligned}$$

```
# 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 (2), the optimal decision rules are

$$\begin{aligned} c_t^* &= y_t - \beta^2 a_t - (1-\beta)\, b_t, \\ b_{t+1}^* &= \beta^{-1} c_t^* + \beta^{-1} b_t - \beta^{-1} y_t \\ &= \beta^{-1} y_t - \beta a_t - \left(\beta^{-1} - 1\right) b_t + \beta^{-1} b_t - \beta^{-1} y_t \\ &= b_t - \beta a_t. \end{aligned}$$

Now we construct two Linear State Space models that emerge from using optimal policies $u_t = -Fx_t$ for the control variable.

Take the original representation case 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 of the innovations representation case, 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)
```

In the following we compute the impulse response functions of c_t and b_t.

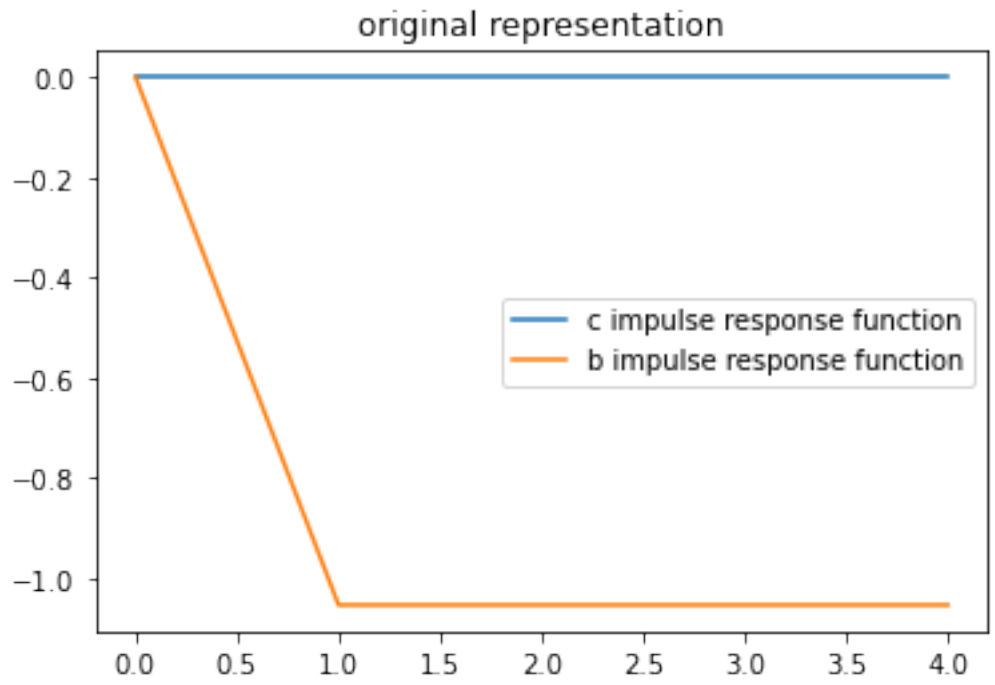
```
J = 5 # Number of coefficients that we want

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([1.99997796e-11, 1.89473992e-11, 1.78947621e-11, 1.68421319e-11,
        1.57894947e-11]),
 array([ 0.        , -1.05263158, -1.05263158, -1.05263158, -1.05263158]))
```

```
plt.title("original 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 0x7f3224670ac0>
```



The above two impulse response functions show that when the consumer has the information assumed in the original representation, 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.

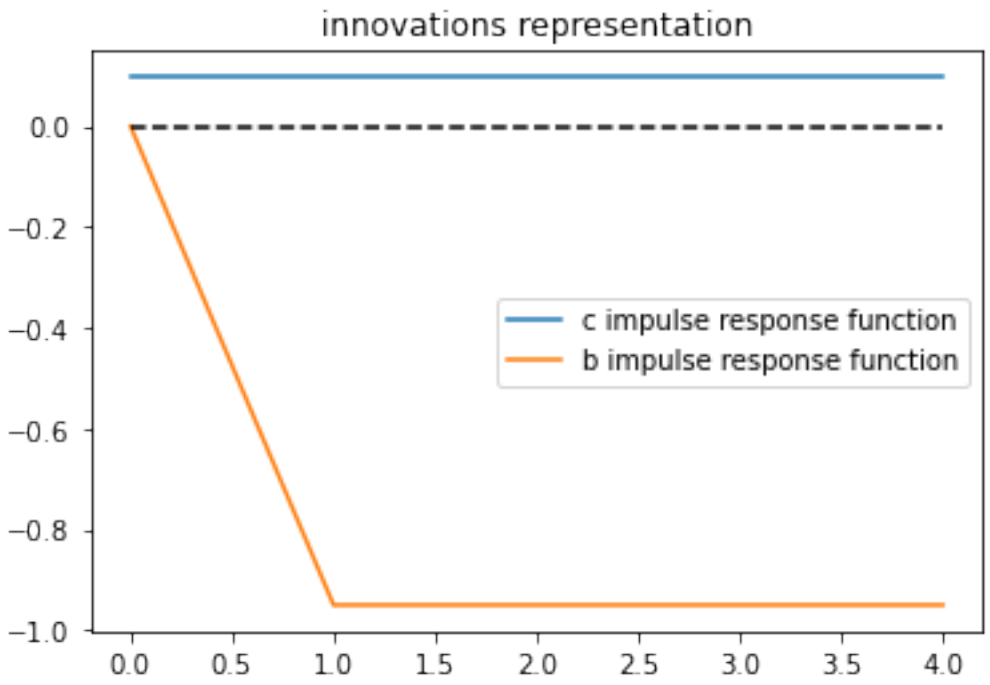
To see this notice, that starting from next period on, his debt permanently **decreases** by β^{-1}

```
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 0x7f32243a5160>
```



The above impulse responses show that when the consumer has only the information that is assumed to be available under the innovations representation 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 in order to lower b_{t+1} 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, constructed as described below.

```
# 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("original representation")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f321ffbce50>
```


```
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 0x7f321ff8efd0>
```

5.3.2 Simulating the Income Process and Two Associated Shock Processes

We now describe how we form a **single** $\{y_t\}_{t=0}^T$ realization that we will use to simulate the two different decision rules 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 **same** $\{y_t\}$ realization generated in step 1 and form an innovation process $\{a_t\}$ from the formulas

$$\begin{aligned} a_0 &= 0 \\ a_t &= \sum_{j=0}^{t-1} \beta^j (y_{t-j} - y_{t-j-1}) + \beta^t a_0, \quad t \geq 1 \end{aligned}$$

3. We throw away the first S observations and form the 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.

5.3.3 Calculating Innovations in Another Way

Here we use formula (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

$$\begin{aligned} a_{t+1} &= \beta a_t + \epsilon_{t+1} - \beta^{-1}\epsilon_t \\ &= \beta\left(\beta a_{t-1} + \epsilon_t - \beta^{-1}\epsilon_{t-1}\right) + \epsilon_{t+1} - \beta^{-1}\epsilon_t \\ &= \beta^2 a_{t-1} + \beta\left(\epsilon_t - \beta^{-1}\epsilon_{t-1}\right) + \epsilon_{t+1} - \beta^{-1}\epsilon_t \\ &= \cdots \\ &= \beta^{t+1} a_0 + \sum_{j=0}^{t} \beta^j \left(\epsilon_{t+1-j} - \beta^{-1}\epsilon_{t-j}\right) \\ &= \beta^{t+1} a_0 + \epsilon_{t+1} + \left(\beta - \beta^{-1}\right) \sum_{j=0}^{t-1} \beta^j \epsilon_{t-j} - \beta^{t-1}\epsilon_0. \end{aligned}$$

We can verify that we recover the same $\{a_t\}$ sequence computed earlier.

5.3.4 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 [LWY13] in their analysis of **fiscal foresight**.

CHAPTER

SIX

CONSUMPTION SMOOTHING WITH COMPLETE AND INCOMPLETE MARKETS

Contents

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

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

6.1 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 [Hal78]

Complete markets allow a consumer to buy or sell claims contingent on all possible states of the world.

Incomplete markets allow a consumer to buy or sell only a limited set of securities, often only a single risk-free security.

Hall [Hal78] 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 maintain 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 its 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 but distinct 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:

```
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.linalg as la
```

6.1.1 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*

6.2 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 purchasing or 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 are 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}.

6.3 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**

$$q_{t+1}(x_{t+1} \,|\, x_t) = \beta\phi(x_{t+1} \,|\, Ax_t, CC') \tag{1}$$

where $\phi(\cdot \,|\, \mu, \Sigma)$ is a multivariate Gaussian distribution with mean vector μ and covariance matrix Σ.

With the pricing kernel $q_{t+1}(x_{t+1} \,|\, x_t)$ in hand, we can price claims to consumption at time $t+1$ consumption that pay off when $x_{t+1} \in S$ at time $t+1$:

$$\int_S q_{t+1}(x_{t+1} \,|\, x_t) dx_{t+1}$$

where S is a subset of $\mathbb{R}^n$.

The price $\int_S q_{t+1}(x_{t+1} \,|\, x_t) dx_{t+1}$ 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 $q(x_{t+1} \,|\, x_t)$.

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 (1), the value at t of $b(x_{t+1})$ is evidently

$$\beta \int b(x_{t+1})\phi(x_{t+1} \,|\, Ax_t, CC')dx_{t+1} = \beta \mathbb{E}_t b_{t+1}$$

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

$$b_t = S_y (I - \beta A)^{-1} x_t - \frac{1}{1-\beta} \bar{c} \tag{2}$$

where $\bar{c}$ satisfies

$$\bar{b}_0 = S_y (I - \beta A)^{-1} x_0 - \frac{1}{1-\beta} \bar{c} \tag{3}$$

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 (3) and the consumer's debt is the fixed function of the state x_t described by (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.

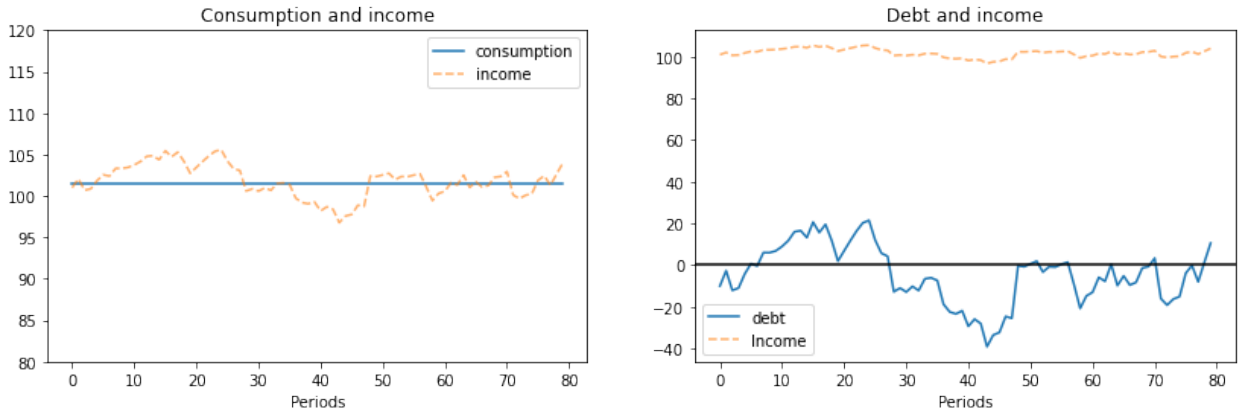
Here's an example that shows how in this setting the availability of insurance against fluctuating nonfinancial income allows the consumer completely to smooth consumption across time and across states of the world

```
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)

```
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()
```



6.3.1 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.

6.3.2 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.

6.3.3 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

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t u(c_t)\right] \quad \text{where} \quad u(c_t) = -(c_t - \gamma)^2 \quad \text{and} \quad 0 < \beta < 1 \tag{4}$$

Here $\gamma > 0$ is a bliss level of consumption

6.3.4 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

6.4 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

$$c_t + b_t \leq y(s_t) + \sum_j q(j \,|\, s_t)\, b_{t+1}(j \,|\, s_t) \tag{5}$$

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

$$q(j \mid i) = \beta P_{ij} \tag{6}$$

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 (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 (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 (6) about Arrow security prices,

$$c_{t+1} = c_t \tag{7}$$

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

$$b_{t+1}(s_{t+1} = j \mid s_t = i) = b(j), \quad i = 1, 2; \;\; j = 1, 2 \tag{8}$$

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 (8) about the consumer's debt level choices.

For $t \geq 1$, these imply

$$\begin{aligned} \bar{c} + b(1) &= y(1) + q(1 \mid 1) b(1) + q(2 \mid 1) b(2) \\ \bar{c} + b(2) &= y(2) + q(1 \mid 2) b(1) + q(2 \mid 2) b(2) \end{aligned} \tag{9}$$

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 the 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

$$\bar{c} + b_0 = y(1) + q(1 \mid 1)b(1) + q(2 \mid 1)b(2) \tag{10}$$

where b_0 is the (exogenous) debt the consumer is assumed to bring into period 0

If we substitute (10) into the first equation of (9) and rearrange, we discover that

$$b(1) = b_0 \tag{11}$$

We can then use the second equation of (9) to deduce the restriction

$$y(1) - y(2) + [q(1 \mid 1) - q(1 \mid 2) - 1]b_0 + [q(2 \mid 1) + 1 - q(2 \mid 2)]b(2) = 0, \tag{12}$$

an equation that we can solve for the unknown $b(2)$.

Knowing $b(1)$ and $b(2)$, we can solve equation (10) for the constant level of consumption $\bar{c}$.

6.4.1 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 satisfied $b(j) = b_0$

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 (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.

6.4.2 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)
    db = ((1 - β) * v - y) / β
```

(continues on next page)

(continued from previous page)

```
    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 [Hal78]

6.5 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

$$\sum_j c_{t+1,j} P_{ij} = c_{t,i} \tag{13}$$

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

$$b_t = \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} - (1-\beta)^{-1} c_t \tag{14}$$

and

$$c_t = (1-\beta) \left[\mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} - b_t \right] \tag{15}$$

Equation (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 (15) into the one-period budget constraint and rearranging leads to

$$b_{t+1} - b_t = \beta^{-1} \left[(1-\beta) \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j y_{t+j} - y_t \right] \tag{16}$$

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 (15), consumption at date t when debt is b_t and the Markov state today is $s_t = i$ is evidently

$$c(b_t, i) = (1-\beta) \left([(I - \beta P)^{-1} \vec{y}]_i - b_t \right) \tag{17}$$

and the increment to debt is

$$b_{t+1} - b_t = \beta^{-1} [(1-\beta) v(i) - y(i)] \tag{18}$$

6.5.1 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.

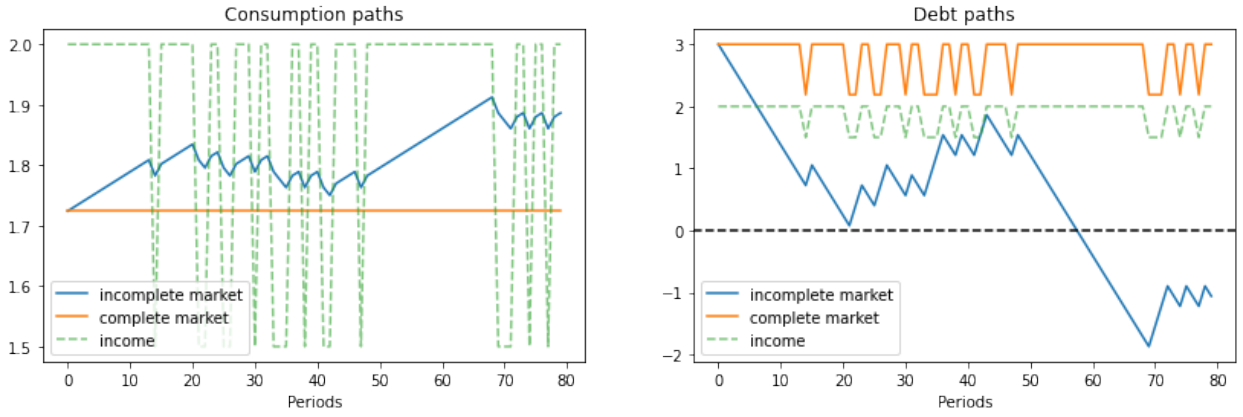
6.5.2 The Incomplete Markets Model

The code above also contains a function called `consumption_incomplete()` that uses (17) and (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.

6.5.3 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 [Bar79] as well as in the complete markets tradition of Lucas and Stokey [LS83].

CHAPTER

SEVEN

TAX SMOOTHING WITH COMPLETE AND INCOMPLETE MARKETS

Contents

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

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

7.1 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 [LS83].
- the other is in the **incomplete markets** tradition of Barro [Bar79].

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 [Bar79] 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 [Hal78] 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 [Bar79] 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 [LS83].

7.1.1 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) [Gal37] seems to have prescribed policies that come from Barro's model [Bar79]

Let's start with some standard imports:

```
import numpy as np
import quantecon as qe
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.linalg as la
```

To exploit the isomorphism between consumption-smoothing and tax-smoothing models, we simply use code from *Consumption Smoothing with Complete and Incomplete Markets*

7.1.2 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

```
class ConsumptionProblem:
    """
    The data for a consumption problem, including some default values.
    """

    def __init__(self,
                 β=.96,
```

(continues on next page)

(continued from previous page)

```
                 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

    associated with the price system

        Q = β * P
    """
    β, P, y, b0, init = cp.β, cp.P, cp.y, cp.b0, cp.init    # Unpack
```

(continues on next page)

(continued from previous page)

```
    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)
    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]
```

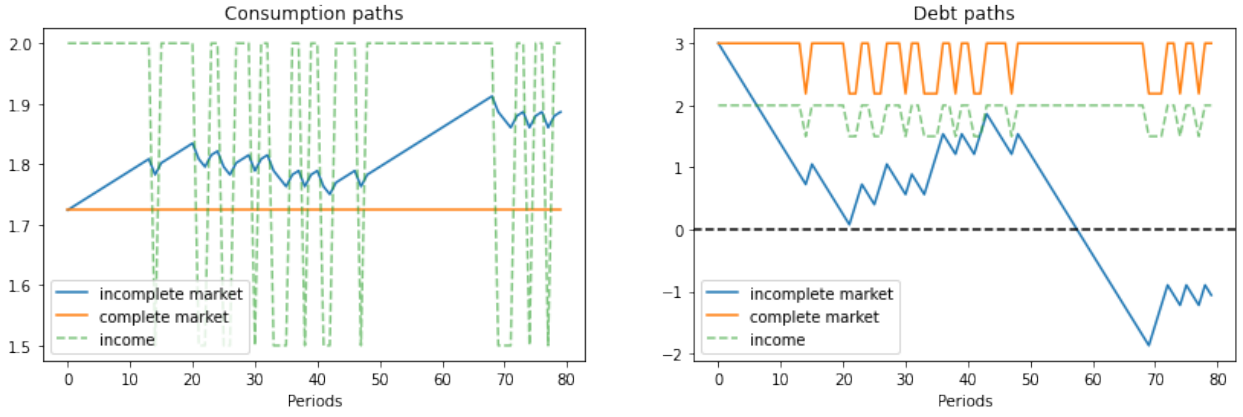
7.1.3 Revisiting the consumption-smoothing model

The code above also contains a function called `consumption_incomplete()` that uses (17) and (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

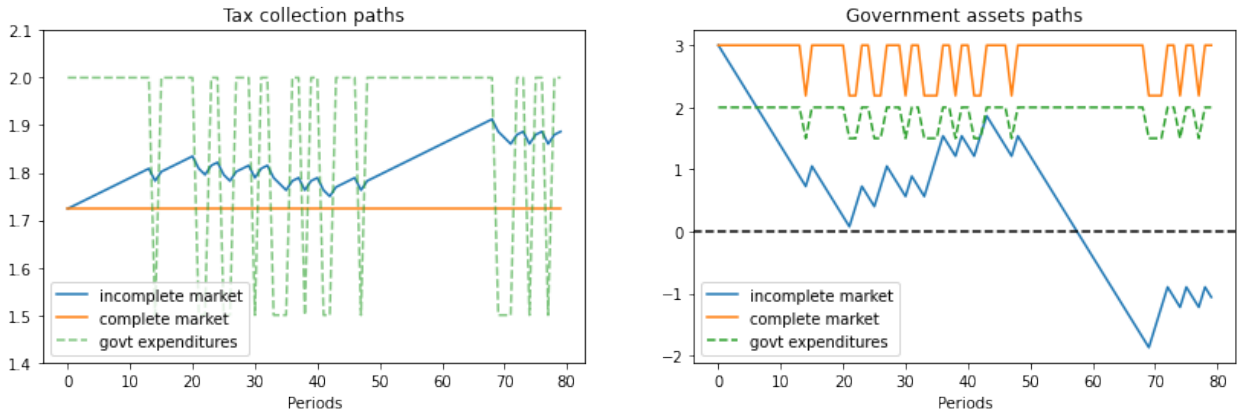
- consumption is constant when there are complete markets.
- consumption takes a random walk in the incomplete markets version of the model.
- the consumer's debt oscillates between two values that are functions of the Markov state in the complete markets model.

- 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()
```



7.2 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 [LS83], 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.

7.3 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.

```
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
```

7.3.1 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)
Q = β * P
```

(continues on next page)

(continued from previous page)

```
# 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)

print("")
print("The cumulative return earned from holding 1 unit market portfolio of␣
↪government bonds")
```

(continues on next page)

(continued from previous page)

```
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
```

7.3.2 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: 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 $b_2 = -10$.

7.4 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.

```
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)
        # 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):
```

(continues on next page)

(continued from previous page)

```
        # 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)[0]
        c1 = line1.get_color()
        ax.set_xlabel('Periods')
        ax.set_ylabel('Cumulative return', color=c1)

        ax_ = ax.twinx()
        ax_._get_lines.prop_cycler = ax._get_lines.prop_cycler
        line2 = ax_.plot(np.arange(N), self.g_path, ls='--')[0]
        c2 = line2.get_color()
        ax_.set_ylabel('Government expenditures', color=c2)

        plt.show()

        # 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}")
```

(continues on next page)

(continued from previous page)

```
        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])
```

7.4.1 Parameters

```
γ = .1
λ = .1
ϕ = .1
θ = .1
ψ = .1
g_L = .5
g_M = .8
g_H = 1.2
β = .96
```

7.4.2 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 $g_L < g_M < g_H$.

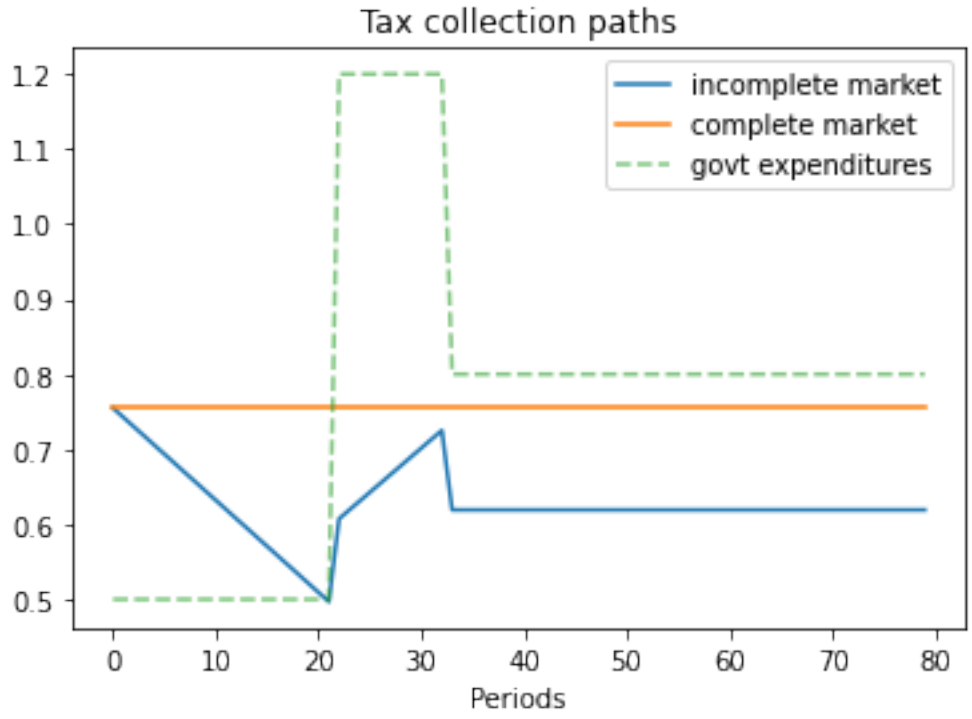
We set $b_0 = 1$ and assume that the initial Markov state is state 1 so that the system starts off in peace.

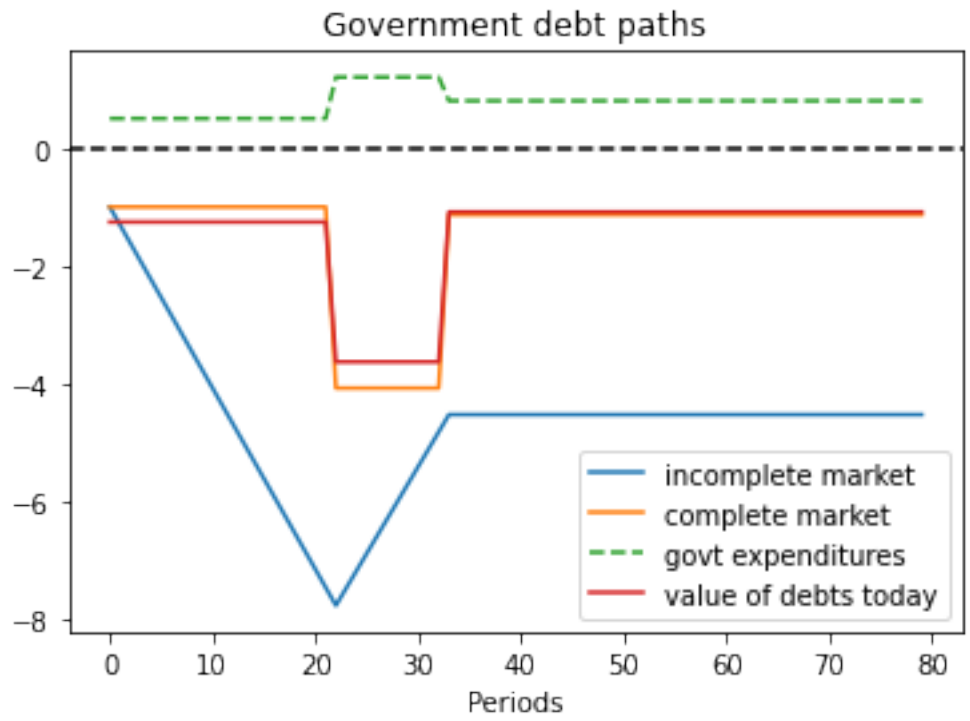
These parameters have government expenditure beginning at a low level, surging during the war, then decreasing after the war to a level that exceeds its prewar level.

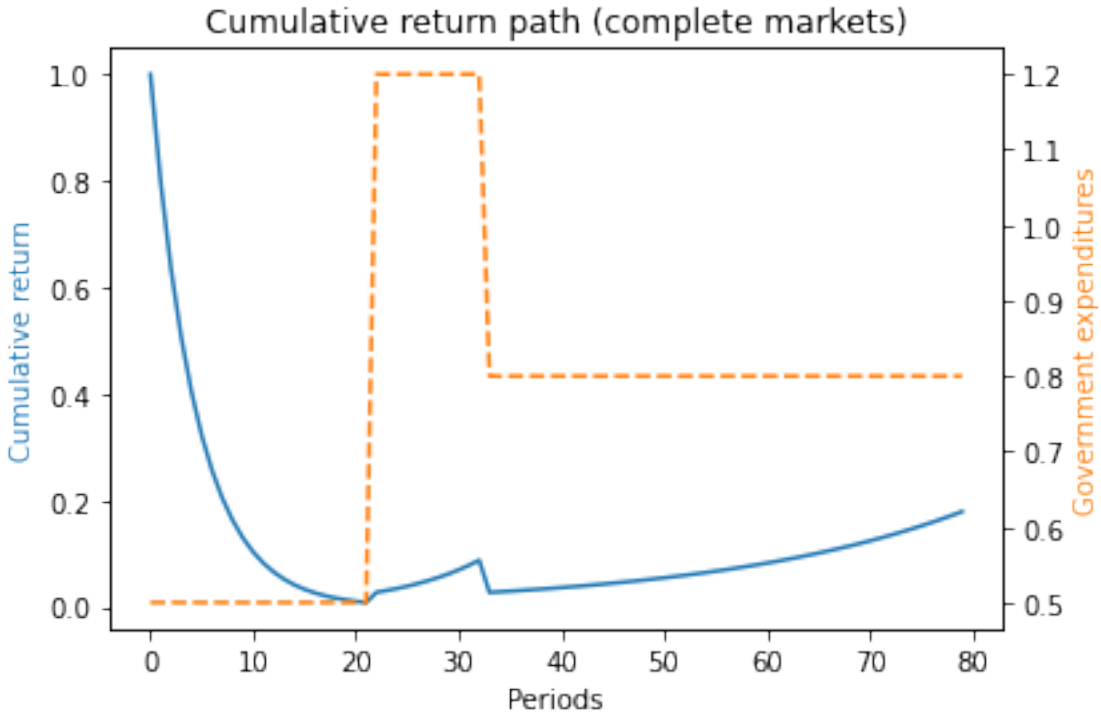
(This type of pattern occurred in the US Civil War and World War I experiences.)

```
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()
```







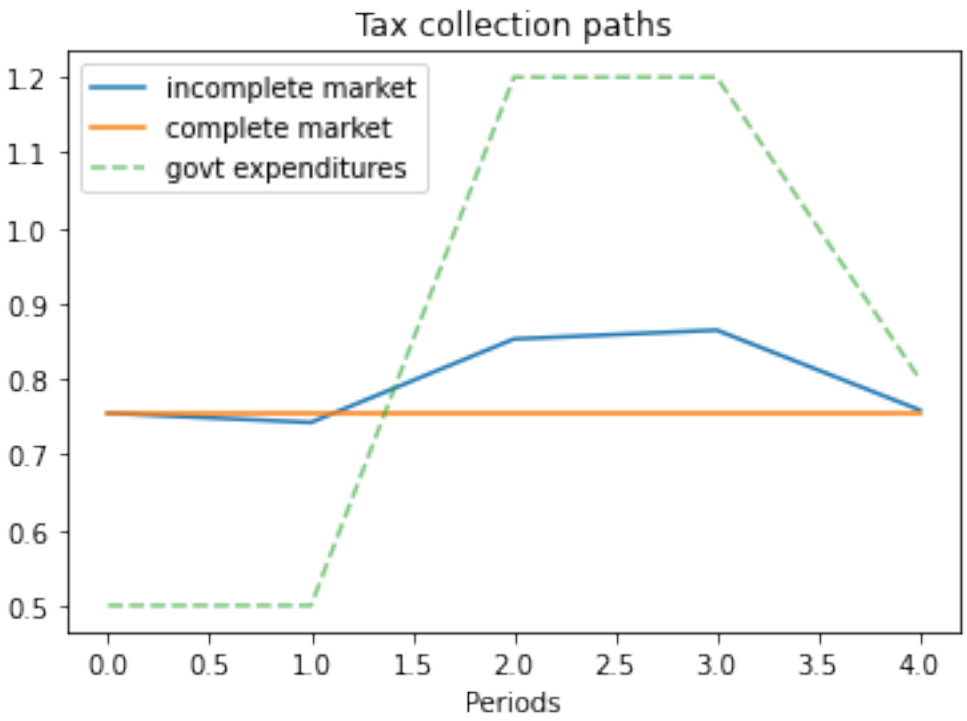

```
P
 [[0.9 0.1 0. ]
 [0.  0.9 0.1]
 [0.  0.  1. ]]
Q
 [[0.864 0.096 0.   ]
```

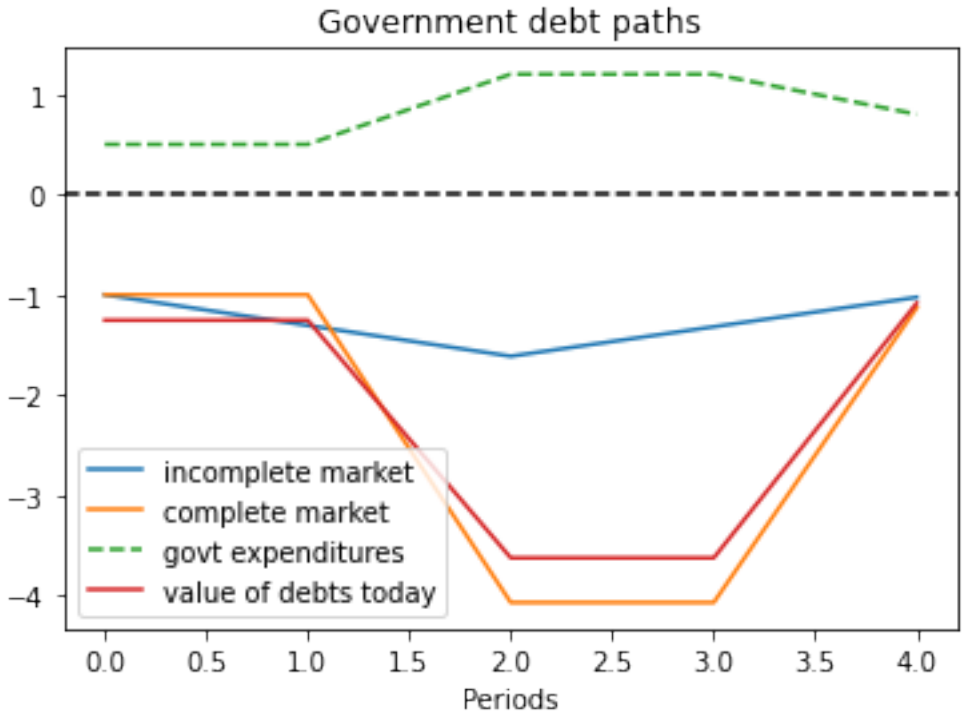
(continues on next page)

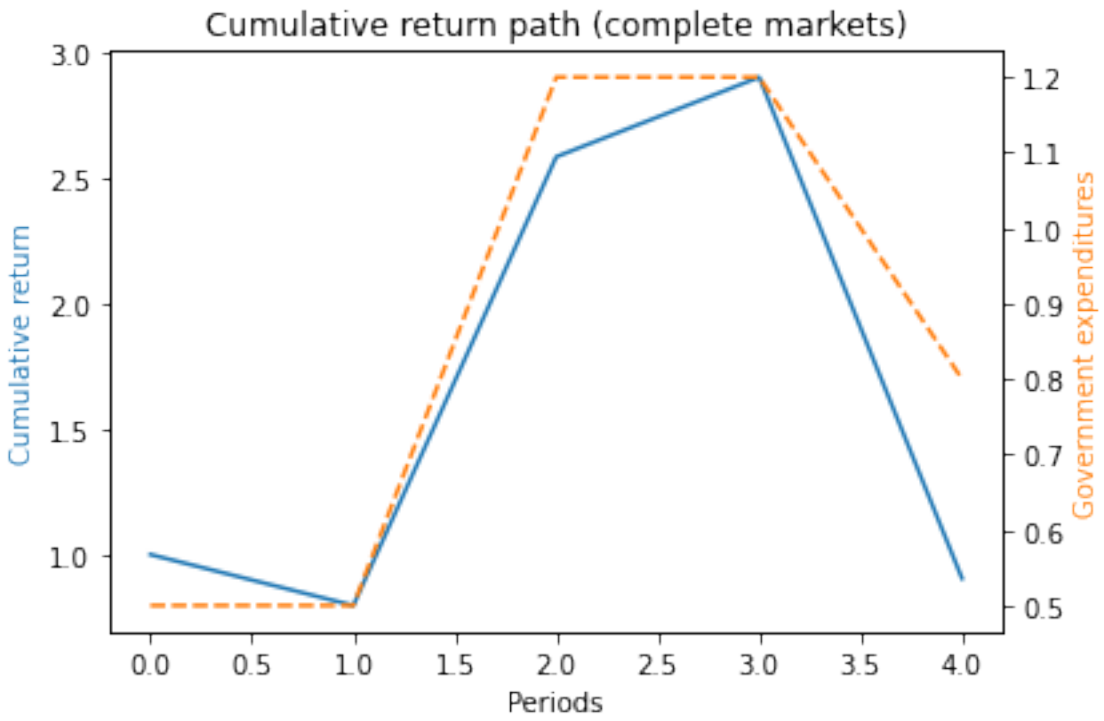
(continued from previous page)

```
 [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 = 1.8845674740484437

Total government spending in peace, war, postwar
  peace = 1.754809688581315
  war = 4.825743944636679
  postwar = 1.8845674740484437

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
  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
0.17908622141460384
```

```
# 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()
```


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
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Periods


Government debt paths
incomplete market
complete market
govt expenditures
value of debts today
1
0
−1
−2
−3
−4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Periods




```
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 = 1.8845674740484437

Total government spending in peace, war, postwar
  peace = 1.754809688581315
  war = 4.825743944636679
  postwar = 1.8845674740484437

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
  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
```

(continues on next page)

(continued from previous page)

```
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
0.9045311615620274
```

7.4.3 Example 2

This example captures a peace followed by a war, eventually followed by a permanent peace .

Here we set

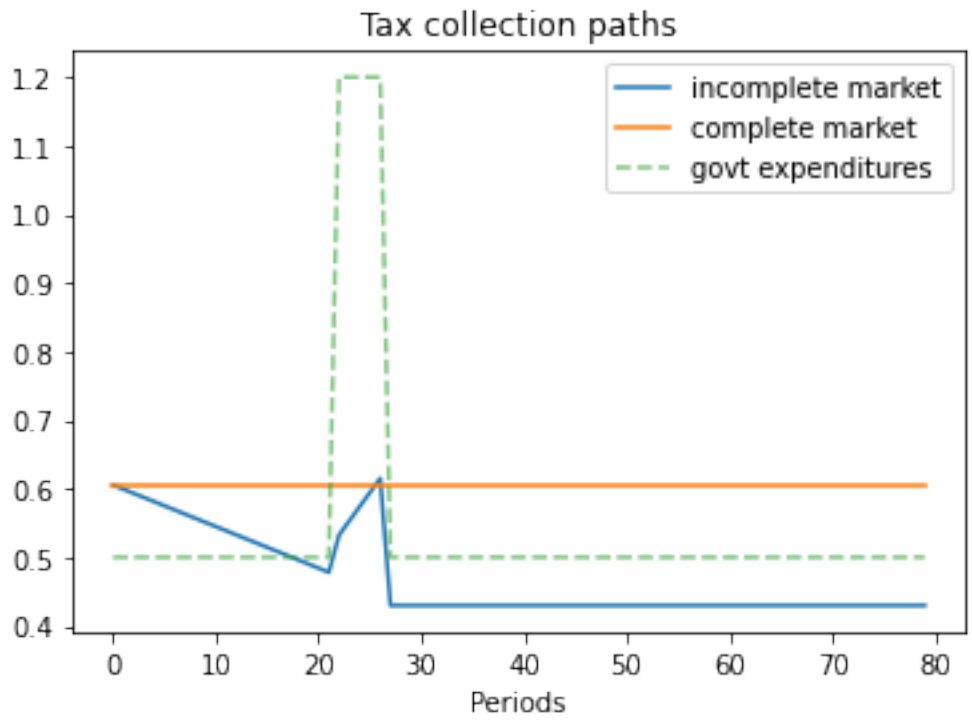
$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-\gamma & \gamma \\ \phi & 0 & 1-\phi \end{bmatrix}$$

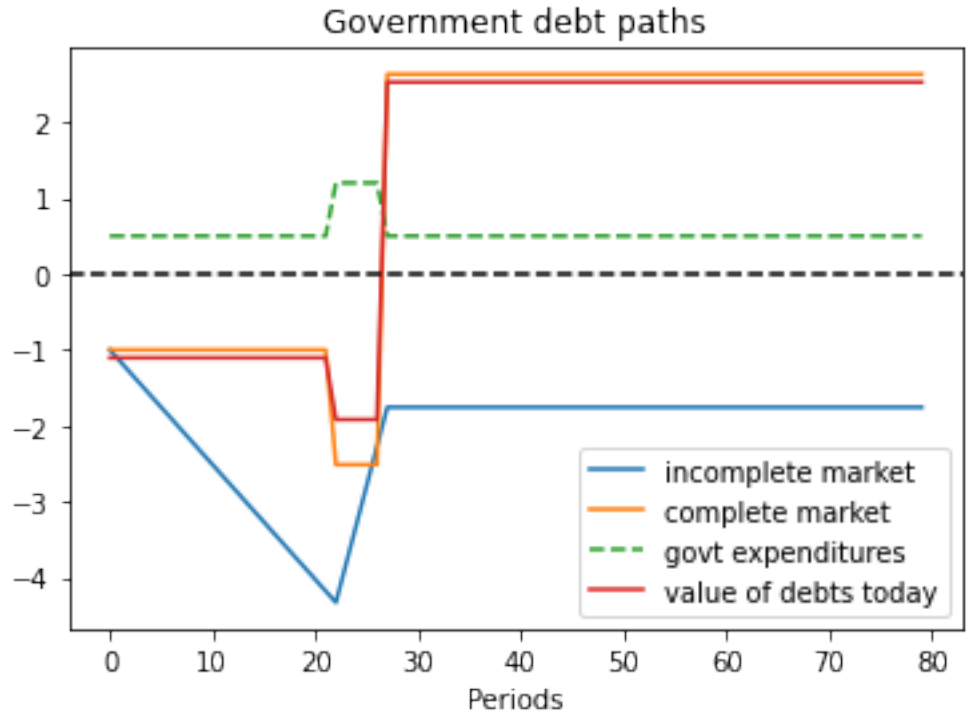
where the government expenditure vector $g = \begin{bmatrix} g_L & g_L & g_H \end{bmatrix}$ and where $g_L < g_H$.

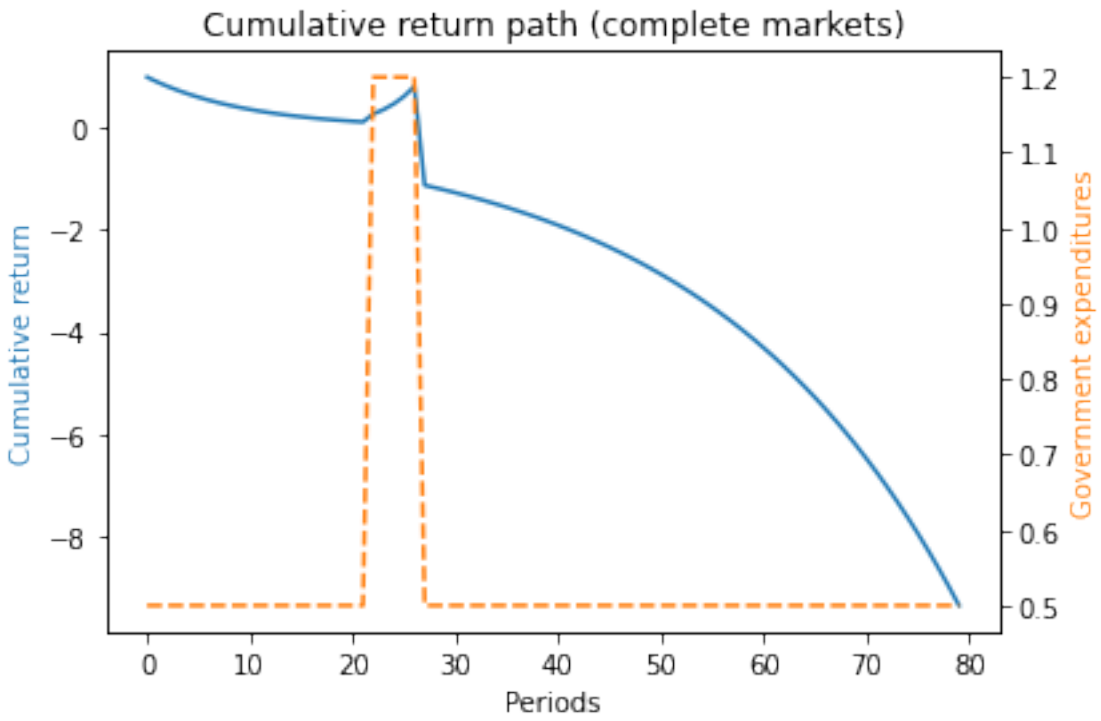
We assume $b_0 = 1$ and that the initial Markov state is state 2 so that the system starts off in a temporary peace.

```
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
incomplete market
complete market
govt expenditures
value of debts today
2
1
0
−1
−2
−3
−4
0
10
20
30
40
50
60
70
80
Periods




```
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 = -2.027889273356399
  T+b in temporary peace = 1.6053287197231834
  T+b in war = 3.1191695501730106

Total government spending in peace, temporary peace, war
  peace = -2.027889273356399
  temporary peace = 1.6053287197231834
  war = 3.119169550173011

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
  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
```

(continues on next page)

(continued from previous page)

```
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
-9.3689917325942
```

7.4.4 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:

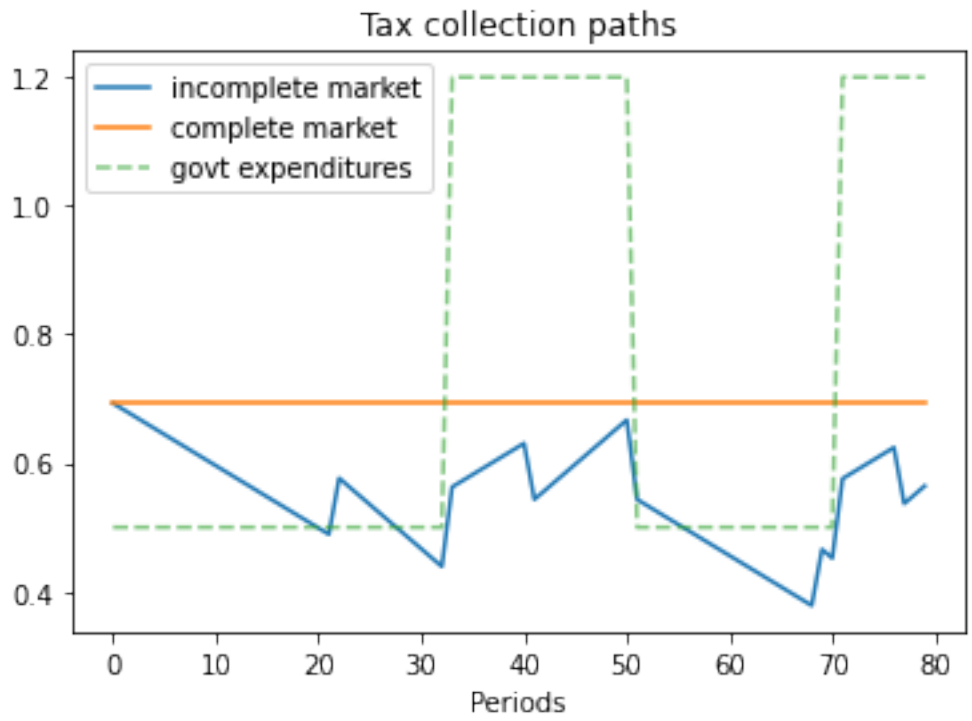
$$P = \begin{bmatrix} 1-\lambda & \lambda & 0 & 0 \\ 0 & 1-\phi & \phi & 0 \\ 0 & 0 & 1-\psi & \psi \\ \theta & 0 & 0 & 1-\theta \end{bmatrix}$$

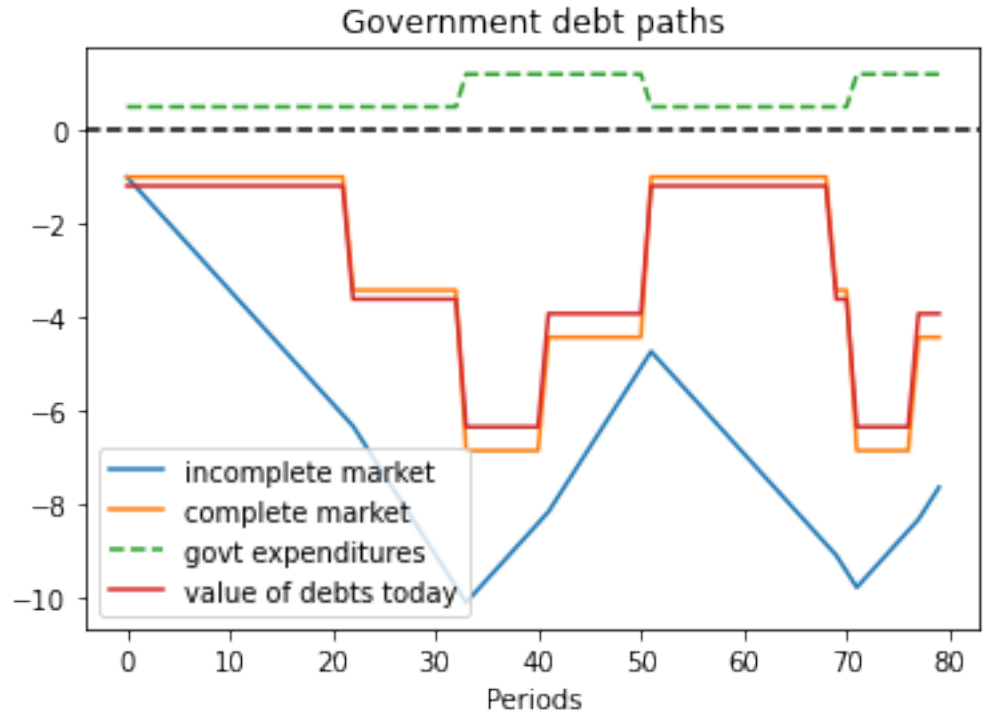
with government expenditure levels for the four states being $\begin{bmatrix} g_L & g_L & g_H & g_H \end{bmatrix}$ where $g_L < g_H$.

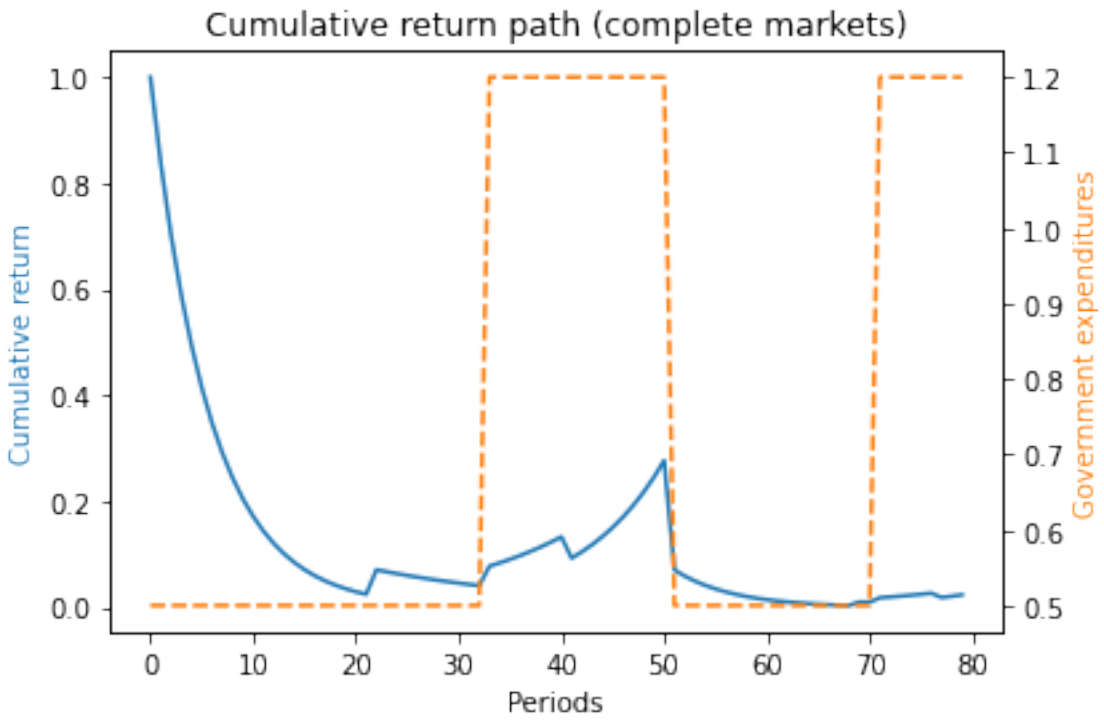
We start with $b_0 = 1$ and $s_0 = 1$.

```
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
incomplete market
complete market
govt expenditures
value of debts today
0
10
20
30
40
50
60
70
80
Periods




```
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 = 5.1281293302540405

Total government spending in peace1, peace2, war1, war2
  peace1 = 1.6927944572748268
  peace2 = 4.117736720554273
  war1 = 7.553071593533487
  war2 = 5.1281293302540405

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
```

(continues on next page)

(continued from previous page)

```
  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
0.023714401788642223
```

7.4.5 Example 4

Here the Markov chain is:

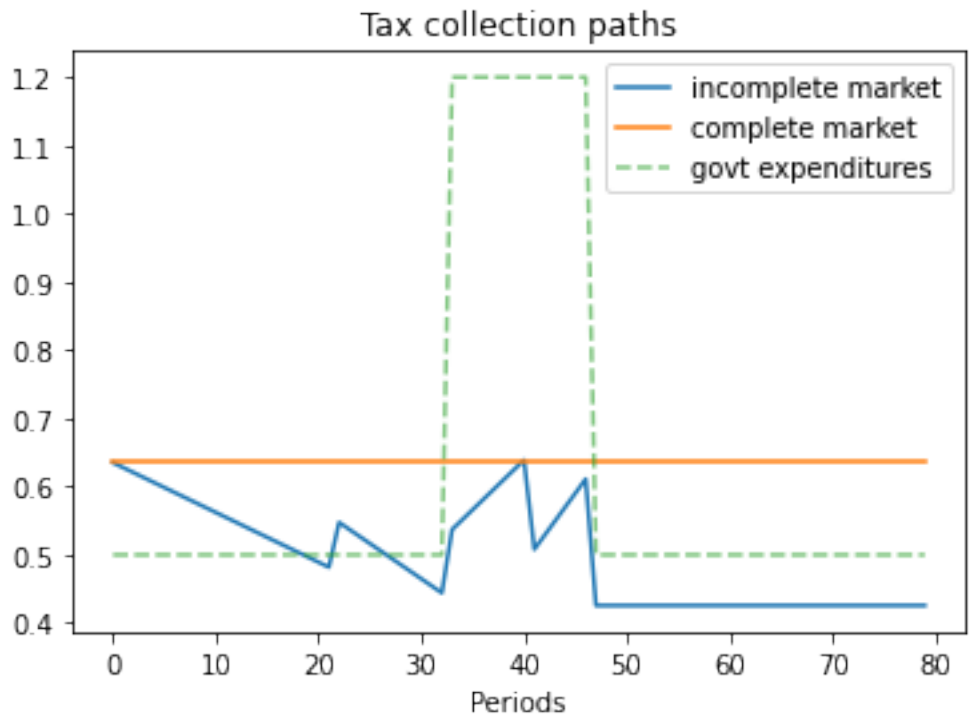
$$P = \begin{bmatrix} 1-\lambda & \lambda & 0 & 0 & 0 \\ 0 & 1-\phi & \phi & 0 & 0 \\ 0 & 0 & 1-\psi & \psi & 0 \\ 0 & 0 & 0 & 1-\theta & \theta \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

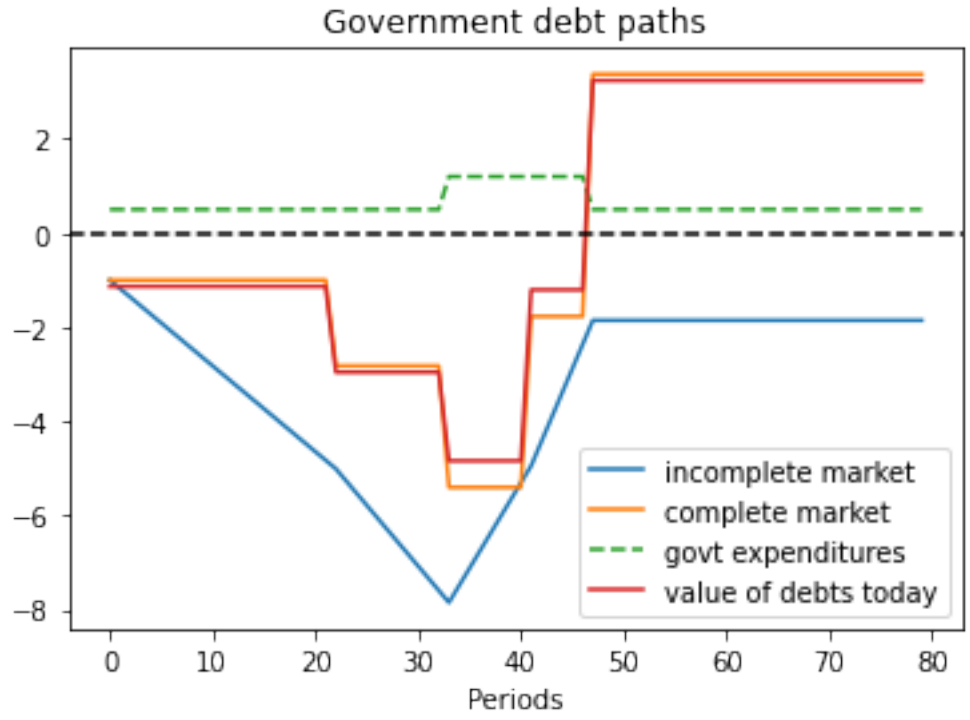
with government expenditure levels for the five states being $\begin{bmatrix} g_L & g_L & g_H & g_H & g_L \end{bmatrix}$ where $g_L < g_H$.

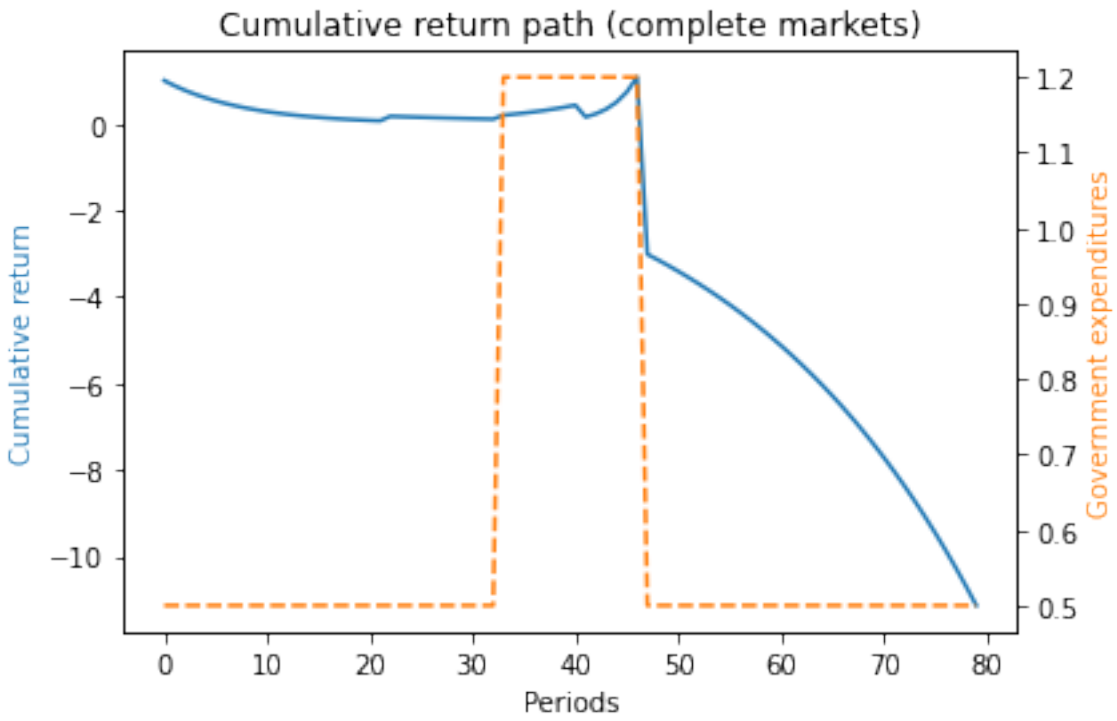
We ssume that $b_0 = 1$ and $s_0 = 1$.

```
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
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
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 = 2.407109110283644
  T+b in permanent peace = -2.739949713245767

Total government spending in peace1, peace2, war1, war2, permanent peace
  peace1 = 1.6349979047185736
  peace2 = 3.457892745537051
  war1 = 6.040327103363228
  war2 = 2.407109110283644
  permanent peace = -2.739949713245767
```

(continues on next page)

(continued from previous page)

```
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
-11.132109773063592
```

7.4.6 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.

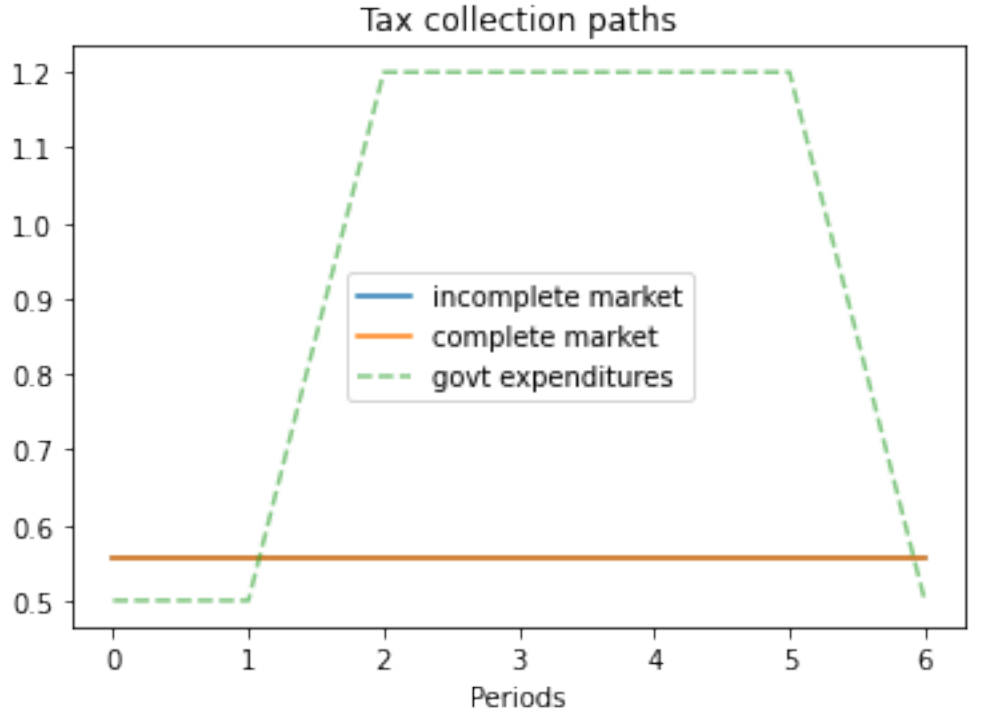
The Markov chain is:

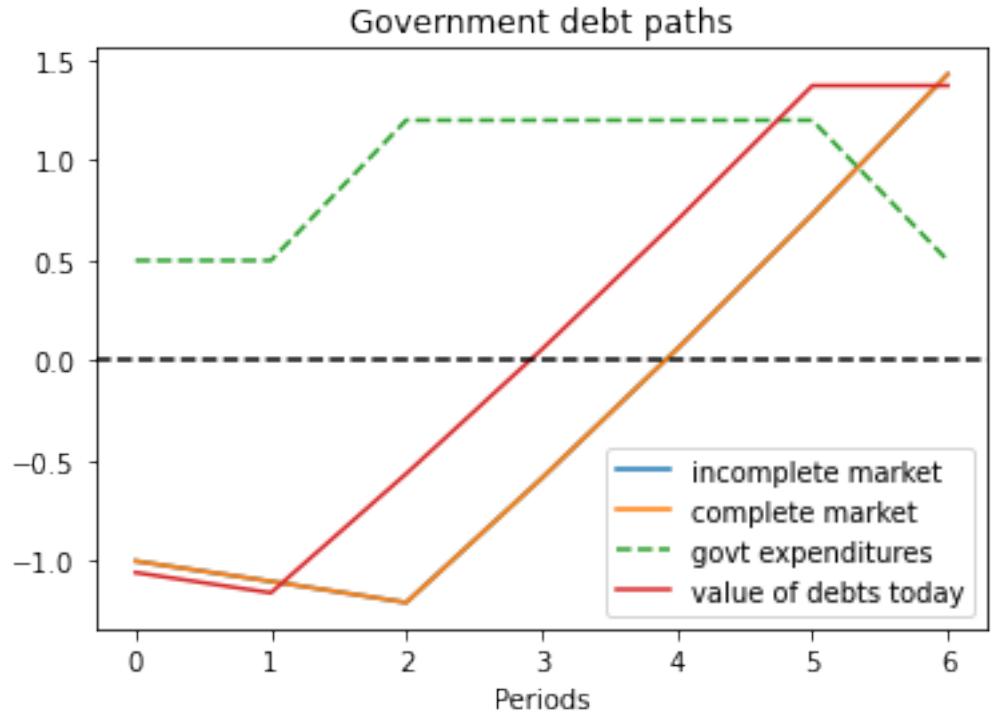
$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

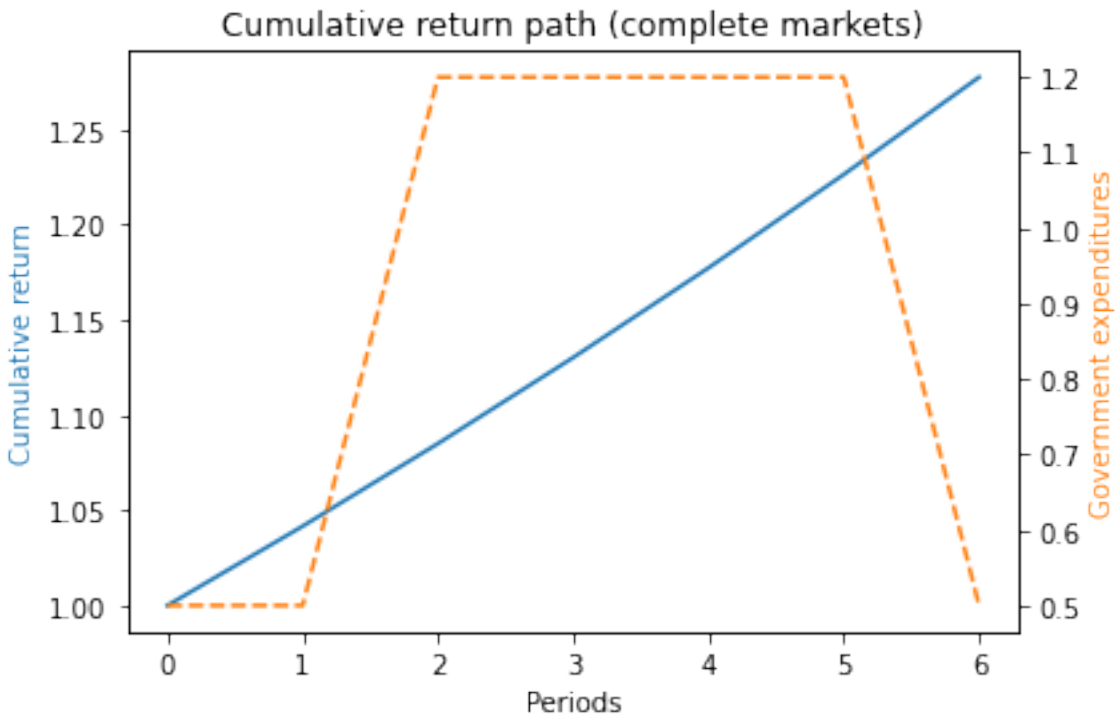
with government expenditure levels for the seven states being $\begin{bmatrix} g_L & g_L & g_H & g_H & g_H & g_H & g_L \end{bmatrix}$ where $g_L < g_H$. Assume $b_0 = 1$ and $s_0 = 1$.

```
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],
                  [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()
```








```
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 = 0.5571895472128002
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 = 1.6584286588928006
 T+b in war1 = 1.7638860668928005
 T+b in war2 = 1.1445708668928007
 T+b in war3 = 0.4994508668928011
 T+b in permanent peace = -0.1725491331071991
```

(continues on next page)

(continued from previous page)

```
Total government spending in peace1, peace2, war1, war2, war3, permanent peace
  peace1 = 1.5571895472128003
  peace2 = 1.6584286588928003
  war1 = 1.7638860668928005
  war2 = 1.1445708668928007
  war3 = 0.4994508668928006
  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
1.2775343959060064
```

7.4.7 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 [Bar79].

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

EIGHT

ROBUSTNESS

Contents

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

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

8.1 Overview

This lecture modifies a Bellman equation to express a decision-maker's doubts about transition dynamics.

His specification doubts make the decision-maker want a *robust* decision rule.

Robust means insensitive to misspecification of transition dynamics.

The decision-maker has a single *approximating model*.

He calls it *approximating* to acknowledge that he doesn't completely trust it.

He fears that outcomes will actually be determined by another model that he cannot describe explicitly.

All that he knows is that the actual data-generating model is in some (uncountable) set of models that surrounds his approximating model.

He quantifies the discrepancy between his approximating model and the genuine data-generating model by using a quantity called *entropy*.

(We'll explain what entropy means below)

He wants a decision rule that will work well enough no matter which of those other models actually governs outcomes.

This is what it means for his decision rule to be "robust to misspecification of an approximating model".

This may sound like too much to ask for, but ....

... a *secret weapon* is available to design robust decision rules.

The secret weapon is max-min control theory.

A value-maximizing decision-maker enlists the aid of an (imaginary) value-minimizing model chooser to construct *bounds* on the value attained by a given decision rule under different models of the transition dynamics.

The original decision-maker uses those bounds to construct a decision rule with an assured performance level, no matter which model actually governs outcomes.

Note: In reading this lecture, please don't think that our decision-maker is paranoid when he conducts a worst-case analysis. By designing a rule that works well against a worst-case, his intention is to construct a rule that will work well across a *set* of models.

Let's start with some imports:

```
import pandas as pd
import numpy as np
from scipy.linalg import eig
import matplotlib.pyplot as plt
%matplotlib inline
import quantecon as qe
```

8.1.1 Sets of Models Imply Sets Of Values

Our "robust" decision-maker wants to know how well a given rule will work when he does not *know* a single transition law ....

... he wants to know *sets* of values that will be attained by a given decision rule F under a *set* of transition laws.

Ultimately, he wants to design a decision rule F that shapes these *sets* of values in ways that he prefers.

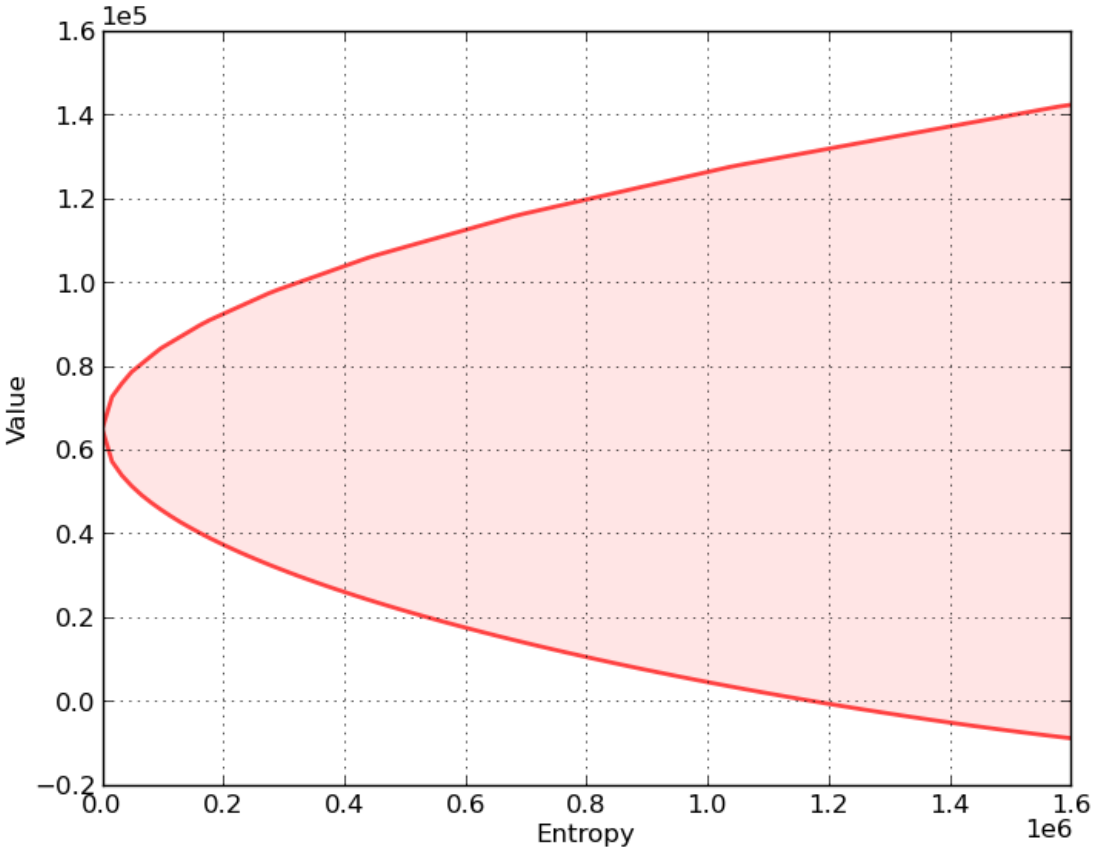
With this in mind, consider the following graph, which relates to a particular decision problem to be explained below

The figure shows a *value-entropy correspondence* for a particular decision rule F.

The shaded set is the graph of the correspondence, which maps entropy to a set of values associated with a set of models that surround the decision-maker's approximating model.

Here

- *Value* refers to a sum of discounted rewards obtained by applying the decision rule F when the state starts at some fixed initial state x_0.
- *Entropy* is a non-negative number that measures the size of a set of models surrounding the decision-maker's approximating model.
  - Entropy is zero when the set includes only the approximating model, indicating that the decision-maker completely trusts the approximating model.
  - Entropy is bigger, and the set of surrounding models is bigger, the less the decision-maker trusts the approximating model.

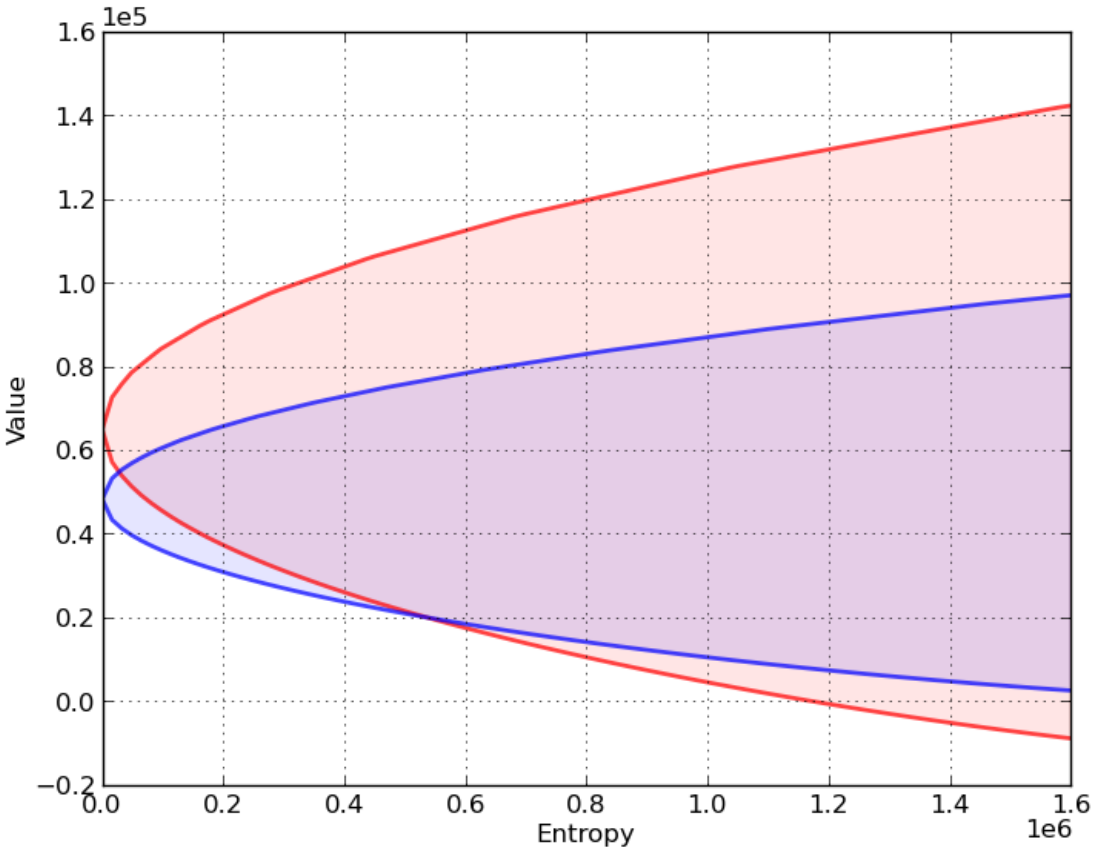

1e5
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
Value
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1e6
Entropy

The shaded region indicates that for **all** models having entropy less than or equal to the number on the horizontal axis, the value obtained will be somewhere within the indicated set of values.

Now let's compare sets of values associated with two different decision rules, F_r and F_b.

In the next figure,

- The red set shows the value-entropy correspondence for decision rule F_r.
- The blue set shows the value-entropy correspondence for decision rule F_b.



The blue correspondence is skinnier than the red correspondence.

This conveys the sense in which the decision rule F_b is *more robust* than the decision rule F_r

- *more robust* means that the set of values is less sensitive to *increasing misspecification* as measured by entropy

Notice that the less robust rule F_r promises higher values for small misspecifications (small entropy).

(But it is more fragile in the sense that it is more sensitive to perturbations of the approximating model)

Below we'll explain in detail how to construct these sets of values for a given F, but for now ….

Here is a hint about the *secret weapons* we'll use to construct these sets

- We'll use some min problems to construct the lower bounds
- We'll use some max problems to construct the upper bounds

We will also describe how to choose F to shape the sets of values.

This will involve crafting a *skinnier* set at the cost of a lower *level* (at least for low values of entropy).

8.1.2 Inspiring Video

If you want to understand more about why one serious quantitative researcher is interested in this approach, we recommend Lars Peter Hansen's Nobel lecture.

8.1.3 Other References

Our discussion in this lecture is based on

- [HS00]
- [HS08a]

8.2 The Model

For simplicity, we present ideas in the context of a class of problems with linear transition laws and quadratic objective functions.

To fit in with our earlier lecture on LQ control, we will treat loss minimization rather than value maximization.

To begin, recall the infinite horizon LQ problem, where an agent chooses a sequence of controls $\{u_t\}$ to minimize

$$\sum_{t=0}^{\infty} \beta^t \left\{x_t' R x_t + u_t' Q u_t\right\} \tag{1}$$

subject to the linear law of motion

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

As before,

- 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 $n \times j$
- R is $n \times n$ and Q is $k \times k$

Here x_t is the state, u_t is the control, and w_t is a shock vector.

For now, we take $\{w_t\} := \{w_t\}_{t=1}^{\infty}$ to be deterministic — a single fixed sequence.

We also allow for *model uncertainty* on the part of the agent solving this optimization problem.

In particular, the agent takes $w_t = 0$ for all $t \geq 0$ as a benchmark model but admits the possibility that this model might be wrong.

As a consequence, she also considers a set of alternative models expressed in terms of sequences $\{w_t\}$ that are "close" to the zero sequence.

She seeks a policy that will do well enough for a set of alternative models whose members are pinned down by sequences $\{w_t\}$.

Soon we'll quantify the quality of a model specification in terms of the maximal size of the expression $\sum_{t=0}^{\infty} \beta^{t+1} w_{t+1}' w_{t+1}$.

8.3 Constructing More Robust Policies

If our agent takes $\{w_t\}$ as a given deterministic sequence, then, drawing on intuition from earlier lectures on dynamic programming, we can anticipate Bellman equations such as

$$J_{t-1}(x) = \min_u \{x'Rx + u'Qu + \beta J_t(Ax + Bu + Cw_t)\}$$

(Here J depends on t because the sequence $\{w_t\}$ is not recursive)

Our tool for studying robustness is to construct a rule that works well even if an adverse sequence $\{w_t\}$ occurs.

In our framework, "adverse" means "loss increasing".

As we'll see, this will eventually lead us to construct the Bellman equation

$$J(x) = \min_u \max_w \{x'Rx + u'Qu + \beta [J(Ax + Bu + Cw) - \theta w'w]\} \tag{3}$$

Notice that we've added the penalty term $-\theta w'w$.

Since $w'w = \|w\|^2$, this term becomes influential when w moves away from the origin.

The penalty parameter θ controls how much we penalize the maximizing agent for "harming" the minimizing agent.

By raising θ more and more, we more and more limit the ability of maximizing agent to distort outcomes relative to the approximating model.

So bigger θ is implicitly associated with smaller distortion sequences $\{w_t\}$.

8.3.1 Analyzing the Bellman Equation

So what does J in (3) look like?

As with the ordinary LQ control model, J takes the form $J(x) = x'Px$ for some symmetric positive definite matrix P.

One of our main tasks will be to analyze and compute the matrix P.

Related tasks will be to study associated feedback rules for u_t and w_{t+1}.

First, using matrix calculus, you will be able to verify that

$$\begin{aligned} \max_w \{(Ax + Bu + Cw)'P(Ax + Bu + Cw) - \theta w'w\} \\ = (Ax + Bu)'\mathcal{D}(P)(Ax + Bu) \end{aligned} \tag{4}$$

where

$$\mathcal{D}(P) := P + PC(\theta I - C'PC)^{-1}C'P \tag{5}$$

and I is a $j \times j$ identity matrix. Substituting this expression for the maximum into (3) yields

$$x'Px = \min_u \{x'Rx + u'Qu + \beta (Ax + Bu)'\mathcal{D}(P)(Ax + Bu)\} \tag{6}$$

Using similar mathematics, the solution to this minimization problem is $u = -Fx$ where $F := (Q + \beta B'\mathcal{D}(P)B)^{-1}\beta B'\mathcal{D}(P)A$.

Substituting this minimizer back into (6) and working through the algebra gives $x'Px = x'\mathcal{B}(\mathcal{D}(P))x$ for all x, or, equivalently,

$$P = \mathcal{B}(\mathcal{D}(P))$$

where $\mathcal{D}$ is the operator defined in (5) and

$$\mathcal{B}(P) := R - \beta^2 A'PB(Q + \beta B'PB)^{-1}B'PA + \beta A'PA$$

The operator $\mathcal{B}$ is the standard (i.e., non-robust) LQ Bellman operator, and $P = \mathcal{B}(P)$ is the standard matrix Riccati equation coming from the Bellman equation — see this discussion.

Under some regularity conditions (see [HS08a]), the operator $\mathcal{B} \circ \mathcal{D}$ has a unique positive definite fixed point, which we denote below by $\hat{P}$.

A robust policy, indexed by θ, is $u = -\hat{F}x$ where

$$\hat{F} := (Q + \beta B'\mathcal{D}(\hat{P})B)^{-1}\beta B'\mathcal{D}(\hat{P})A \tag{7}$$

We also define

$$\hat{K} := (\theta I - C'\hat{P}C)^{-1}C'\hat{P}(A - B\hat{F}) \tag{8}$$

The interpretation of $\hat{K}$ is that $w_{t+1} = \hat{K}x_t$ on the worst-case path of $\{x_t\}$, in the sense that this vector is the maximizer of (4) evaluated at the fixed rule $u = -\hat{F}x$.

Note that $\hat{P}, \hat{F}, \hat{K}$ are all determined by the primitives and θ.

Note also that if θ is very large, then $\mathcal{D}$ is approximately equal to the identity mapping.

Hence, when θ is large, $\hat{P}$ and $\hat{F}$ are approximately equal to their standard LQ values.

Furthermore, when θ is large, $\hat{K}$ is approximately equal to zero.

Conversely, smaller θ is associated with greater fear of model misspecification and greater concern for robustness.

8.4 Robustness as Outcome of a Two-Person Zero-Sum Game

What we have done above can be interpreted in terms of a two-person zero-sum game in which $\hat{F}, \hat{K}$ are Nash equilibrium objects.

Agent 1 is our original agent, who seeks to minimize loss in the LQ program while admitting the possibility of misspecification.

Agent 2 is an imaginary malevolent player.

Agent 2's malevolence helps the original agent to compute bounds on his value function across a set of models.

We begin with agent 2's problem.

8.4.1 Agent 2's Problem

Agent 2

1. knows a fixed policy F specifying the behavior of agent 1, in the sense that $u_t = -Fx_t$ for all t
2. responds by choosing a shock sequence $\{w_t\}$ from a set of paths sufficiently close to the benchmark sequence $\{0, 0, 0, ...\}$

A natural way to say "sufficiently close to the zero sequence" is to restrict the summed inner product $\sum_{t=1}^{\infty} w_t' w_t$ to be small.

However, to obtain a time-invariant recursive formulation, it turns out to be convenient to restrict a discounted inner product

$$\sum_{t=1}^{\infty} \beta^t w_t' w_t \leq \eta \tag{9}$$

Now let F be a fixed policy, and let $J_F(x_0, \mathbf{w})$ be the present-value cost of that policy given sequence $\mathbf{w} := \{w_t\}$ and initial condition $x_0 \in \mathbb{R}^n$.

Substituting $-Fx_t$ for u_t in (1), this value can be written as

$$J_F(x_0, \mathbf{w}) := \sum_{t=0}^{\infty} \beta^t x_t' (R + F'QF) x_t \tag{10}$$

where

$$x_{t+1} = (A - BF)x_t + Cw_{t+1} \tag{11}$$

and the initial condition x_0 is as specified in the left side of (10).

Agent 2 chooses $\mathbf{w}$ to maximize agent 1's loss $J_F(x_0, \mathbf{w})$ subject to (9).

Using a Lagrangian formulation, we can express this problem as

$$\max_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{ x_t' (R + F'QF) x_t - \beta\theta (w_{t+1}' w_{t+1} - \eta) \right\}$$

where $\{x_t\}$ satisfied (11) and θ is a Lagrange multiplier on constraint (9).

For the moment, let's take θ as fixed, allowing us to drop the constant $\beta\theta\eta$ term in the objective function, and hence write the problem as

$$\max_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{ x_t' (R + F'QF) x_t - \beta\theta w_{t+1}' w_{t+1} \right\}$$

or, equivalently,

$$\min_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{ -x_t' (R + F'QF) x_t + \beta\theta w_{t+1}' w_{t+1} \right\} \tag{12}$$

subject to (11).

What's striking about this optimization problem is that it is once again an LQ discounted dynamic programming problem, with $\mathbf{w} = \{w_t\}$ as the sequence of controls.

The expression for the optimal policy can be found by applying the usual LQ formula (see here).

We denote it by $K(F, \theta)$, with the interpretation $w_{t+1} = K(F, \theta) x_t$.

The remaining step for agent 2's problem is to set θ to enforce the constraint (9), which can be done by choosing $\theta = \theta_\eta$ such that

$$\beta \sum_{t=0}^{\infty} \beta^t x_t' K(F, \theta_\eta)' K(F, \theta_\eta) x_t = \eta \tag{13}$$

Here x_t is given by (11) — which in this case becomes $x_{t+1} = (A - BF + CK(F, \theta))x_t$.

8.4.2 Using Agent 2's Problem to Construct Bounds on the Value Sets

The Lower Bound

Define the minimized object on the right side of problem (12) as $R_\theta(x_0, F)$.

Because "minimizers minimize" we have

$$R_\theta(x_0, F) \leq \sum_{t=0}^{\infty} \beta^t \left\{-x_t'(R + F'QF)x_t\right\} + \beta\theta \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1},$$

where $x_{t+1} = (A - BF + CK(F, \theta))x_t$ and x_0 is a given initial condition.

This inequality in turn implies the inequality

$$R_\theta(x_0, F) - \theta \text{ ent} \leq \sum_{t=0}^{\infty} \beta^t \left\{-x_t'(R + F'QF)x_t\right\} \tag{14}$$

where

$$\text{ent} := \beta \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1}$$

The left side of inequality (14) is a straight line with slope $-\theta$.

Technically, it is a "separating hyperplane".

At a particular value of entropy, the line is tangent to the lower bound of values as a function of entropy.

In particular, the lower bound on the left side of (14) is attained when

$$\text{ent} = \beta \sum_{t=0}^{\infty} \beta^t x_t' K(F, \theta)' K(F, \theta) x_t \tag{15}$$

To construct the *lower bound* on the set of values associated with all perturbations **w** satisfying the entropy constraint (9) at a given entropy level, we proceed as follows:

- For a given θ, solve the minimization problem (12).
- Compute the minimizer $R_\theta(x_0, F)$ and the associated entropy using (15).
- Compute the lower bound on the value function $R_\theta(x_0, F) - \theta$ ent and plot it against ent.
- Repeat the preceding three steps for a range of values of θ to trace out the lower bound.

Note: This procedure sweeps out a set of separating hyperplanes indexed by different values for the Lagrange multiplier θ.

The Upper Bound

To construct an *upper bound* we use a very similar procedure.

We simply replace the *minimization* problem (12) with the *maximization* problem

$$V_{\tilde{\theta}}(x_0, F) = \max_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{-x_t'(R + F'QF)x_t - \beta\tilde{\theta} w_{t+1}' w_{t+1}\right\} \tag{16}$$

where now $\tilde{\theta} > 0$ penalizes the choice of **w** with larger entropy.

(Notice that $\tilde{\theta} = -\theta$ in problem (12))

Because "maximizers maximize" we have

$$V_{\tilde{\theta}}(x_0, F) \geq \sum_{t=0}^{\infty} \beta^t \left\{-x_t'(R + F'QF)x_t\right\} - \beta\tilde{\theta} \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1}$$

which in turn implies the inequality

$$V_{\tilde{\theta}}(x_0, F) + \tilde{\theta}\,\text{ent} \geq \sum_{t=0}^{\infty} \beta^t \left\{-x_t'(R + F'QF)x_t\right\} \tag{17}$$

where

$$\text{ent} \equiv \beta \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1}$$

The left side of inequality (17) is a straight line with slope $\tilde{\theta}$.

The upper bound on the left side of (17) is attained when

$$\text{ent} = \beta \sum_{t=0}^{\infty} \beta^t x_t' K(F, \tilde{\theta})' K(F, \tilde{\theta}) x_t \tag{18}$$

To construct the *upper bound* on the set of values associated all perturbations **w** with a given entropy we proceed much as we did for the lower bound

- For a given $\tilde{\theta}$, solve the maximization problem (16).
- Compute the maximizer $V_{\tilde{\theta}}(x_0, F)$ and the associated entropy using (18).
- Compute the upper bound on the value function $V_{\tilde{\theta}}(x_0, F) + \tilde{\theta}$ ent and plot it against ent.
- Repeat the preceding three steps for a range of values of $\tilde{\theta}$ to trace out the upper bound.

Reshaping the Set of Values

Now in the interest of *reshaping* these sets of values by choosing F, we turn to agent 1's problem.

8.4.3 Agent 1's Problem

Now we turn to agent 1, who solves

$$\min_{\{u_t\}} \sum_{t=0}^{\infty} \beta^t \left\{x_t' R x_t + u_t' Q u_t - \beta\theta w_{t+1}' w_{t+1}\right\} \tag{19}$$

where $\{w_{t+1}\}$ satisfies $w_{t+1} = Kx_t$.

In other words, agent 1 minimizes

$$\sum_{t=0}^{\infty} \beta^t \left\{x_t'(R - \beta\theta K'K)x_t + u_t' Q u_t\right\} \tag{20}$$

subject to

$$x_{t+1} = (A + CK)x_t + Bu_t \tag{21}$$

Once again, the expression for the optimal policy can be found here — we denote it by $\tilde{F}$.

8.4.4 Nash Equilibrium

Clearly, the $\tilde{F}$ we have obtained depends on K, which, in agent 2's problem, depended on an initial policy F.

Holding all other parameters fixed, we can represent this relationship as a mapping Φ, where

$$\tilde{F} = \Phi(K(F, \theta))$$

The map $F \mapsto \Phi(K(F, \theta))$ corresponds to a situation in which

1. agent 1 uses an arbitrary initial policy F
2. agent 2 best responds to agent 1 by choosing $K(F, \theta)$
3. agent 1 best responds to agent 2 by choosing $\tilde{F} = \Phi(K(F, \theta))$

As you may have already guessed, the robust policy $\hat{F}$ defined in (7) is a fixed point of the mapping Φ.

In particular, for any given θ,

1. $K(\hat{F}, \theta) = \hat{K}$, where $\hat{K}$ is as given in (8)
2. $\Phi(\hat{K}) = \hat{F}$

A sketch of the proof is given in *the appendix*.

8.5 The Stochastic Case

Now we turn to the stochastic case, where the sequence $\{w_t\}$ is treated as an IID sequence of random vectors.

In this setting, we suppose that our agent is uncertain about the *conditional probability distribution* of w_{t+1}.

The agent takes the standard normal distribution $N(0, I)$ as the baseline conditional distribution, while admitting the possibility that other "nearby" distributions prevail.

These alternative conditional distributions of w_{t+1} might depend nonlinearly on the history x_s, $s \leq t$.

To implement this idea, we need a notion of what it means for one distribution to be near another one.

Here we adopt a very useful measure of closeness for distributions known as the *relative entropy*, or Kullback-Leibler divergence.

For densities p, q, the Kullback-Leibler divergence of q from p is defined as

$$D_{KL}(p, q) := \int \ln \left[\frac{p(x)}{q(x)} \right] p(x) \, dx$$

Using this notation, we replace (3) with the stochastic analog

$$J(x) = \min_u \max_{\psi \in \mathcal{P}} \left\{ x' R x + u' Q u + \beta \left[\int J(Ax + Bu + Cw) \, \psi(dw) - \theta D_{KL}(\psi, \phi) \right] \right\} \tag{22}$$

Here $\mathcal{P}$ represents the set of all densities on $\mathbb{R}^n$ and ϕ is the benchmark distribution $N(0, I)$.

The distribution ϕ is chosen as the least desirable conditional distribution in terms of next period outcomes, while taking into account the penalty term $\theta D_{KL}(\psi, \phi)$.

This penalty term plays a role analogous to the one played by the deterministic penalty $\theta w' w$ in (3), since it discourages large deviations from the benchmark.

8.5.1 Solving the Model

The maximization problem in (22) appears highly nontrivial — after all, we are maximizing over an infinite dimensional space consisting of the entire set of densities.

However, it turns out that the solution is tractable, and in fact also falls within the class of normal distributions.

First, we note that J has the form $J(x) = x'Px + d$ for some positive definite matrix P and constant real number d.

Moreover, it turns out that if $(I - \theta^{-1}C'PC)^{-1}$ is nonsingular, then

$$\begin{aligned} \max_{\psi \in \mathcal{P}} \left\{ \int (Ax + Bu + Cw)'P(Ax + Bu + Cw)\,\psi(dw) - \theta D_{KL}(\psi, \phi) \right\} \\ = (Ax + Bu)'\mathcal{D}(P)(Ax + Bu) + \kappa(\theta, P) \end{aligned} \tag{23}$$

where

$$\kappa(\theta, P) := \theta \ln[\det(I - \theta^{-1}C'PC)^{-1}]$$

and the maximizer is the Gaussian distribution

$$\psi = N\left((\theta I - C'PC)^{-1}C'P(Ax + Bu), (I - \theta^{-1}C'PC)^{-1}\right) \tag{24}$$

Substituting the expression for the maximum into Bellman equation (22) and using $J(x) = x'Px + d$ gives

$$x'Px + d = \min_u \left\{ x'Rx + u'Qu + \beta\,(Ax + Bu)'\mathcal{D}(P)(Ax + Bu) + \beta\,[d + \kappa(\theta, P)] \right\} \tag{25}$$

Since constant terms do not affect minimizers, the solution is the same as (6), leading to

$$x'Px + d = x'\mathcal{B}(\mathcal{D}(P))x + \beta\,[d + \kappa(\theta, P)]$$

To solve this Bellman equation, we take $\hat{P}$ to be the positive definite fixed point of $\mathcal{B} \circ \mathcal{D}$.

In addition, we take $\hat{d}$ as the real number solving $d = \beta\,[d + \kappa(\theta, P)]$, which is

$$\hat{d} := \frac{\beta}{1 - \beta}\kappa(\theta, P) \tag{26}$$

The robust policy in this stochastic case is the minimizer in (25), which is once again $u = -\hat{F}x$ for $\hat{F}$ given by (7).

Substituting the robust policy into (24) we obtain the worst-case shock distribution:

$$w_{t+1} \sim N(\hat{K}x_t, (I - \theta^{-1}C'\hat{P}C)^{-1})$$

where $\hat{K}$ is given by (8).

Note that the mean of the worst-case shock distribution is equal to the same worst-case w_{t+1} as in the earlier deterministic setting.

8.5.2 Computing Other Quantities

Before turning to implementation, we briefly outline how to compute several other quantities of interest.

Worst-Case Value of a Policy

One thing we will be interested in doing is holding a policy fixed and computing the discounted loss associated with that policy.

So let F be a given policy and let $J_F(x)$ be the associated loss, which, by analogy with (22), satisfies

$$J_F(x) = \max_{\psi \in \mathcal{P}} \left\{ x'(R + F'QF)x + \beta \left[\int J_F((A - BF)x + Cw)\, \psi(dw) - \theta D_{KL}(\psi, \phi) \right] \right\}$$

Writing $J_F(x) = x' P_F x + d_F$ and applying the same argument used to derive (23) we get

$$x' P_F x + d_F = x'(R + F'QF)x + \beta \left[x'(A - BF)' \mathcal{D}(P_F)(A - BF)x + d_F + \kappa(\theta, P_F) \right]$$

To solve this we take P_F to be the fixed point

$$P_F = R + F'QF + \beta(A - BF)' \mathcal{D}(P_F)(A - BF)$$

and

$$d_F := \frac{\beta}{1 - \beta} \kappa(\theta, P_F) = \frac{\beta}{1 - \beta} \theta \ln[\det(I - \theta^{-1} C' P_F C)^{-1}] \tag{27}$$

If you skip ahead to *the appendix*, you will be able to verify that $-P_F$ is the solution to the Bellman equation in agent 2's problem *discussed above* — we use this in our computations.

8.6 Implementation

The QuantEcon.py package provides a class called `RBLQ` for implementation of robust LQ optimal control.

The code can be found on GitHub.

Here is a brief description of the methods of the class

- `d_operator()` and `b_operator()` implement $\mathcal{D}$ and $\mathcal{B}$ respectively
- `robust_rule()` and `robust_rule_simple()` both solve for the triple $\hat{F}, \hat{K}, \hat{P}$, as described in equations (7) – (8) and the surrounding discussion
  - `robust_rule()` is more efficient
  - `robust_rule_simple()` is more transparent and easier to follow
- `K_to_F()` and `F_to_K()` solve the decision problems of *agent 1* and *agent 2* respectively
- `compute_deterministic_entropy()` computes the left-hand side of (13)
- `evaluate_F()` computes the loss and entropy associated with a given policy — see *this discussion*

8.7 Application

Let us consider a monopolist similar to this one, but now facing model uncertainty.

The inverse demand function is $p_t = a_0 - a_1 y_t + d_t$.

where

$$d_{t+1} = \rho d_t + \sigma_d w_{t+1}, \quad \{w_t\} \overset{\text{IID}}{\sim} N(0, 1)$$

and all parameters are strictly positive.

The period return function for the monopolist is

$$r_t = p_t y_t - \gamma \frac{(y_{t+1} - y_t)^2}{2} - c y_t$$

Its objective is to maximize expected discounted profits, or, equivalently, to minimize $\mathbb{E} \sum_{t=0}^{\infty} \beta^t (-r_t)$.

To form a linear regulator problem, we take the state and control to be

$$x_t = \begin{bmatrix} 1 \\ y_t \\ d_t \end{bmatrix} \quad \text{and} \quad u_t = y_{t+1} - y_t$$

Setting $b := (a_0 - c)/2$ we define

$$R = - \begin{bmatrix} 0 & b & 0 \\ b & -a_1 & 1/2 \\ 0 & 1/2 & 0 \end{bmatrix} \quad \text{and} \quad Q = \gamma/2$$

For the transition matrices, we set

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \rho \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad C = \begin{bmatrix} 0 \\ 0 \\ \sigma_d \end{bmatrix}$$

Our aim is to compute the value-entropy correspondences *shown above*.

The parameters are

$$a_0 = 100, a_1 = 0.5, \rho = 0.9, \sigma_d = 0.05, \beta = 0.95, c = 2, \gamma = 50.0$$

The standard normal distribution for w_t is understood as the agent's baseline, with uncertainty parameterized by θ.

We compute value-entropy correspondences for two policies

1. The no concern for robustness policy F_0, which is the ordinary LQ loss minimizer.
2. A "moderate" concern for robustness policy F_b, with $\theta = 0.02$.

The code for producing the graph shown above, with blue being for the robust policy, is as follows

```
# Model parameters

a_0 = 100
a_1 = 0.5
ρ = 0.9
σ_d = 0.05
β = 0.95
c = 2
γ = 50.0

θ = 0.002
ac = (a_0 - c) / 2.0

# Define LQ matrices

R = np.array([[0.,    ac,    0.],
              [ac, -a_1,   0.5],
              [0.,   0.5,   0.]])
```

(continues on next page)

(continued from previous page)

```
R = -R  # For minimization
Q = γ / 2

A = np.array([[1., 0., 0.],
              [0., 1., 0.],
              [0., 0., ρ]])
B = np.array([[0.],
              [1.],
              [0.]])
C = np.array([[0.],
              [0.],
              [σ_d]])

# ------------------------------------------------------------------------ #
#                                  Functions                               #
# ------------------------------------------------------------------------ #


def evaluate_policy(θ, F):

    """
    Given θ (scalar, dtype=float) and policy F (array_like), returns the
    value associated with that policy under the worst case path for {w_t},
    as well as the entropy level.
    """

    rlq = qe.robustlq.RBLQ(Q, R, A, B, C, β, θ)
    K_F, P_F, d_F, O_F, o_F = rlq.evaluate_F(F)
    x0 = np.array([[1.], [0.], [0.]])
    value = - x0.T @ P_F @ x0 - d_F
    entropy = x0.T @ O_F @ x0 + o_F
    return list(map(float, (value, entropy)))


def value_and_entropy(emax, F, bw, grid_size=1000):

    """
    Compute the value function and entropy levels for a θ path
    increasing until it reaches the specified target entropy value.

    Parameters
    ==========
    emax: scalar
        The target entropy value

    F: array_like
        The policy function to be evaluated

    bw: str
        A string specifying whether the implied shock path follows best
        or worst assumptions. The only acceptable values are 'best' and
        'worst'.

    Returns
    =======
    df: pd.DataFrame
```

(continues on next page)

(continued from previous page)

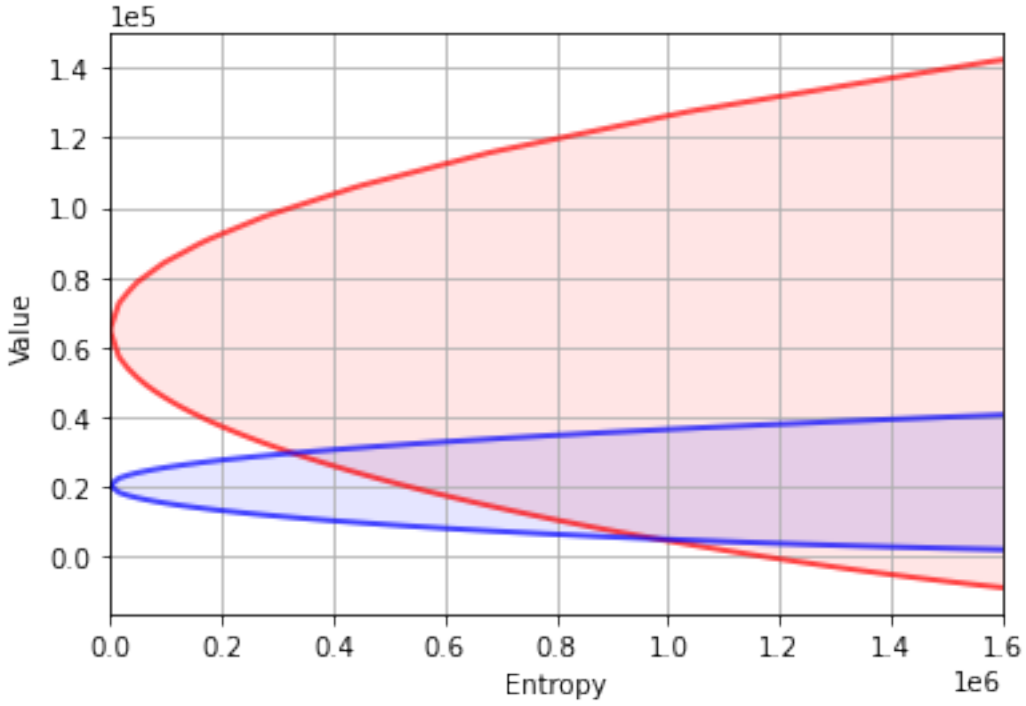
```
        A pandas DataFrame containing the value function and entropy
        values up to the emax parameter. The columns are 'value' and
        'entropy'.
    """

    if bw == 'worst':
        θs = 1 / np.linspace(1e-8, 1000, grid_size)
    else:
        θs = -1 / np.linspace(1e-8, 1000, grid_size)

    df = pd.DataFrame(index=θs, columns=('value', 'entropy'))

    for θ in θs:
        df.loc[θ] = evaluate_policy(θ, F)
        if df.loc[θ, 'entropy'] >= emax:
            break

    df = df.dropna(how='any')
    return df


# ------------------------------------------------------------------------ #
#                                   Main
# ------------------------------------------------------------------------ #


# Compute the optimal rule
optimal_lq = qe.lqcontrol.LQ(Q, R, A, B, C, beta=β)
Po, Fo, do = optimal_lq.stationary_values()

# Compute a robust rule given θ
baseline_robust = qe.robustlq.RBLQ(Q, R, A, B, C, β, θ)
Fb, Kb, Pb = baseline_robust.robust_rule()

# Check the positive definiteness of worst-case covariance matrix to
# ensure that θ exceeds the breakdown point
test_matrix = np.identity(Pb.shape[0]) - (C.T @ Pb @ C) / θ
eigenvals, eigenvecs = eig(test_matrix)
assert (eigenvals >= 0).all(), 'θ below breakdown point.'


emax = 1.6e6

optimal_best_case = value_and_entropy(emax, Fo, 'best')
robust_best_case = value_and_entropy(emax, Fb, 'best')
optimal_worst_case = value_and_entropy(emax, Fo, 'worst')
robust_worst_case = value_and_entropy(emax, Fb, 'worst')

fig, ax = plt.subplots()

ax.set_xlim(0, emax)
ax.set_ylabel("Value")
ax.set_xlabel("Entropy")
ax.grid()

for axis in 'x', 'y':
    plt.ticklabel_format(style='sci', axis=axis, scilimits=(0, 0))
```

(continues on next page)

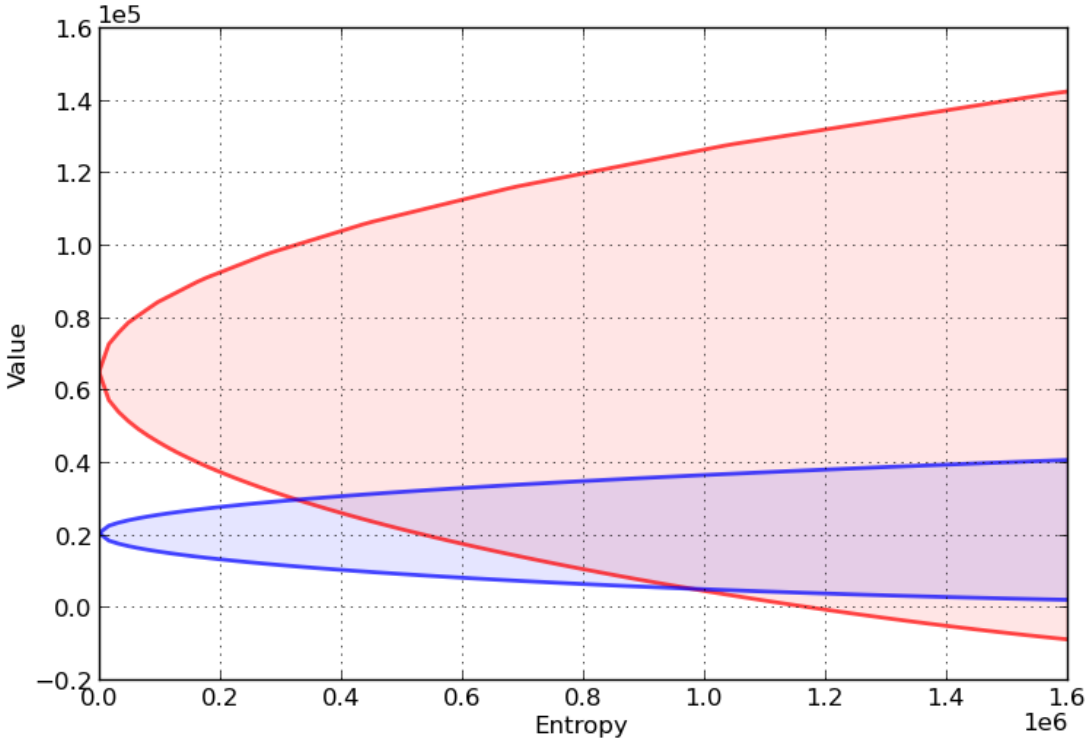
(continued from previous page)

```
plot_args = {'lw': 2, 'alpha': 0.7}

colors = 'r', 'b'

df_pairs = ((optimal_best_case, optimal_worst_case),
            (robust_best_case, robust_worst_case))


class Curve:

    def __init__(self, x, y):
        self.x, self.y = x, y

    def __call__(self, z):
        return np.interp(z, self.x, self.y)


for c, df_pair in zip(colors, df_pairs):
    curves = []
    for df in df_pair:
        # Plot curves
        x, y = df['entropy'], df['value']
        x, y = (np.asarray(a, dtype='float') for a in (x, y))
        egrid = np.linspace(0, emax, 100)
        curve = Curve(x, y)
        print(ax.plot(egrid, curve(egrid), color=c, **plot_args))
        curves.append(curve)
    # Color fill between curves
    ax.fill_between(egrid,
                    curves[0](egrid),
                    curves[1](egrid),
                    color=c, alpha=0.1)

plt.show()
```

```
[<matplotlib.lines.Line2D object at 0x7fc971be28e0>]
[<matplotlib.lines.Line2D object at 0x7fc971bd8b20>]
[<matplotlib.lines.Line2D object at 0x7fc9706a9c10>]
[<matplotlib.lines.Line2D object at 0x7fc9706cf130>]
```



Here's another such figure, with $\theta = 0.002$ instead of 0.02



Can you explain the different shape of the value-entropy correspondence for the robust policy?

8.8 Appendix

We sketch the proof only of the first claim in *this section*, which is that, for any given θ, $K(\hat{F}, \theta) = \hat{K}$, where $\hat{K}$ is as given in (8).

This is the content of the next lemma.

Lemma. If $\hat{P}$ is the fixed point of the map $\mathcal{B} \circ \mathcal{D}$ and $\hat{F}$ is the robust policy as given in (7), then

$$K(\hat{F}, \theta) = (\theta I - C' \hat{P} C)^{-1} C' \hat{P} (A - B\hat{F}) \tag{28}$$

Proof: As a first step, observe that when $F = \hat{F}$, the Bellman equation associated with the LQ problem (11) – (12) is

$$\tilde{P} = -R - \hat{F}' Q \hat{F} - \beta^2 (A - B\hat{F})' \tilde{P} C (\beta\theta I + \beta C' \tilde{P} C)^{-1} C' \tilde{P} (A - B\hat{F}) + \beta (A - B\hat{F})' \tilde{P} (A - B\hat{F}) \tag{29}$$

(revisit this discussion if you don't know where (29) comes from) and the optimal policy is

$$w_{t+1} = -\beta (\beta\theta I + \beta C' \tilde{P} C)^{-1} C' \tilde{P} (A - B\hat{F}) x_t$$

Suppose for a moment that $-\hat{P}$ solves the Bellman equation (29).

In this case, the policy becomes

$$w_{t+1} = (\theta I - C' \hat{P} C)^{-1} C' \hat{P} (A - B\hat{F}) x_t$$

which is exactly the claim in (28).

Hence it remains only to show that $-\hat{P}$ solves (29), or, in other words,

$$\hat{P} = R + \hat{F}' Q \hat{F} + \beta (A - B\hat{F})' \hat{P} C (\theta I - C' \hat{P} C)^{-1} C' \hat{P} (A - B\hat{F}) + \beta (A - B\hat{F})' \hat{P} (A - B\hat{F})$$

Using the definition of $\mathcal{D}$, we can rewrite the right-hand side more simply as

$$R + \hat{F}' Q \hat{F} + \beta (A - B\hat{F})' \mathcal{D}(\hat{P}) (A - B\hat{F})$$

Although it involves a substantial amount of algebra, it can be shown that the latter is just $\hat{P}$.

(Hint: Use the fact that $\hat{P} = \mathcal{B}(\mathcal{D}(\hat{P}))$)

CHAPTER

NINE

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
```

9.1 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 [DVGC99] and the references cited there.

The method has been applied to problems in macroeconomics and monetary economics by [SW+08] and [SW09].

The periodic models of seasonality described in chapter 14 of [HS13] 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.

9.2 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

$$-(x_t' P x_t + \rho)$$

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.

9.3 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 $s_t \in [1, 2, \dots, N]$ 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 $i = 1, \dots, n$ satisfy the N interrelated Bellman equations

$$-x' P_i x - \rho_i = \max_u - \left[x' R_i x + u' Q_i u + 2 u' W_i x + \beta \sum_j \Pi_{ij} E((A_i x + B_i u + C_i w)' P_j (A_i x + B_i u + C_i w) x + \rho_j) \right]$$

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

$$F_i = (Q_i + \beta \sum_j \Pi_{ij} B_i' P_j B_i)^{-1} (\beta \sum_j \Pi_{ij} (B_i' P_j A_i) + W_i)$$

9.4 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

%matplotlib inline
```

```
# Set discount factor
β = 0.95
```

9.5 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

$$\begin{aligned} r\left(s_t, k_t\right) &= f_{1,s_t} k_t - f_{2,s_t} k_t^2 - d_{s_t} {u_t}^2 \\ &= -x_t' \underbrace{\begin{bmatrix} f_{2,s_t} & -\frac{f_{1,s_t}}{2} \\ -\frac{f_{1,s_t}}{2} & 0 \end{bmatrix}}_{\equiv R(s_t)} x_t + \underbrace{d_{s_t}}_{\equiv Q(s_t)} {u_t}^2 \end{aligned}$$

$$x_{t+1} = \begin{bmatrix} k_{t+1} \\ 1 \end{bmatrix} = \underbrace{I_2}_{\equiv A(s_t)} x_t + \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\equiv B(s_t)} u_t$$

```
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.

```
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]]])
```

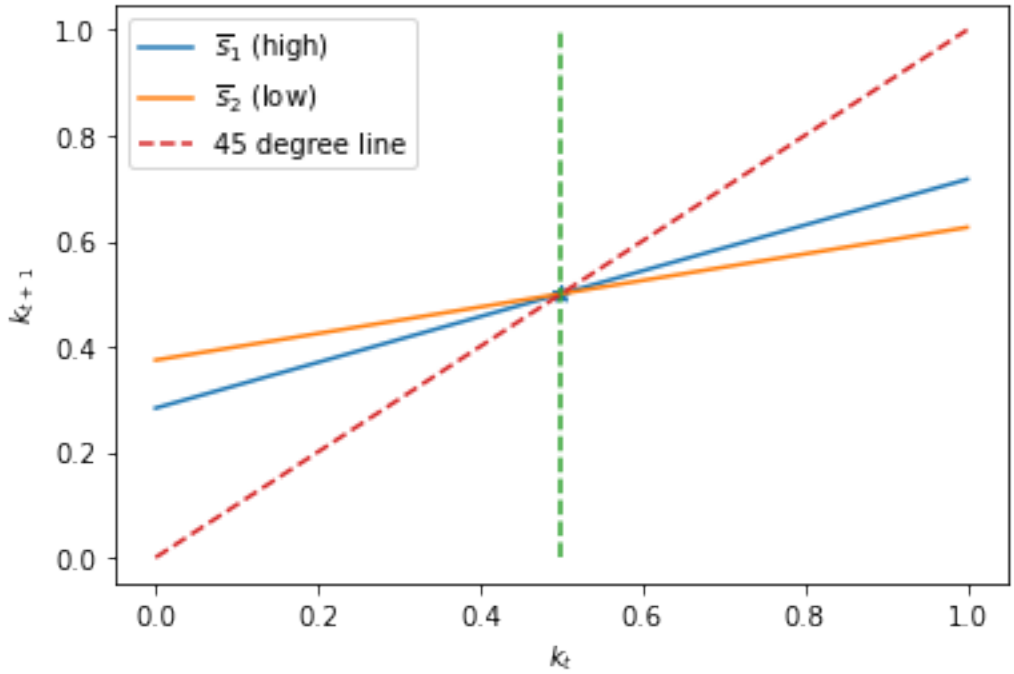
Now we'll plot the decision rules and see if they make sense

```
# 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.

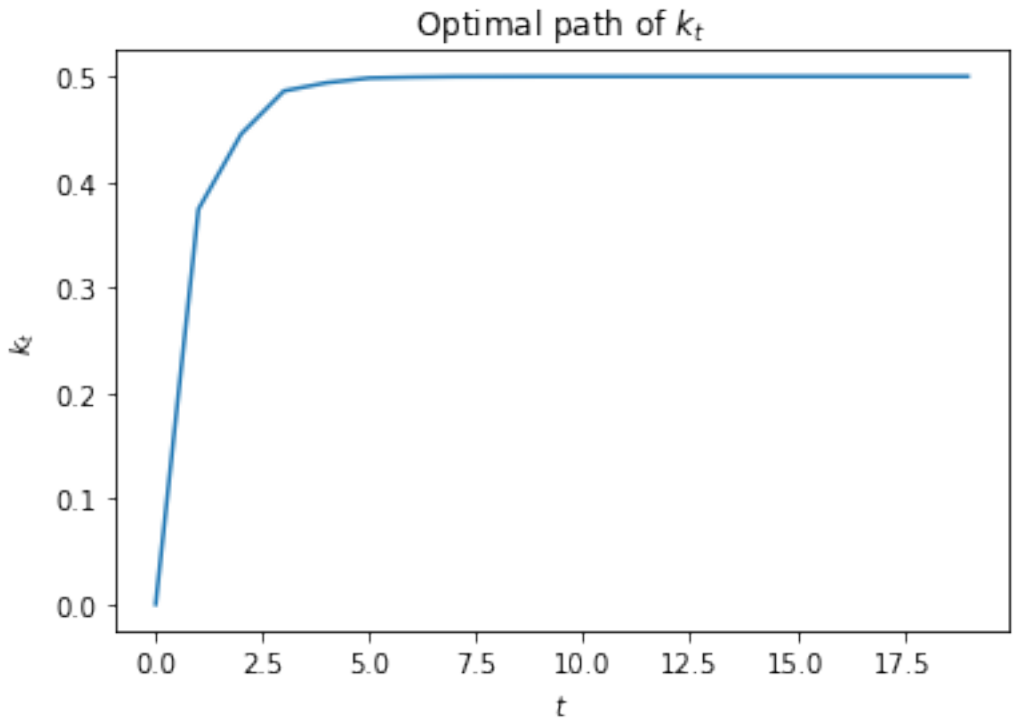
It also plots the 45 degree line.

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]

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]]])
```

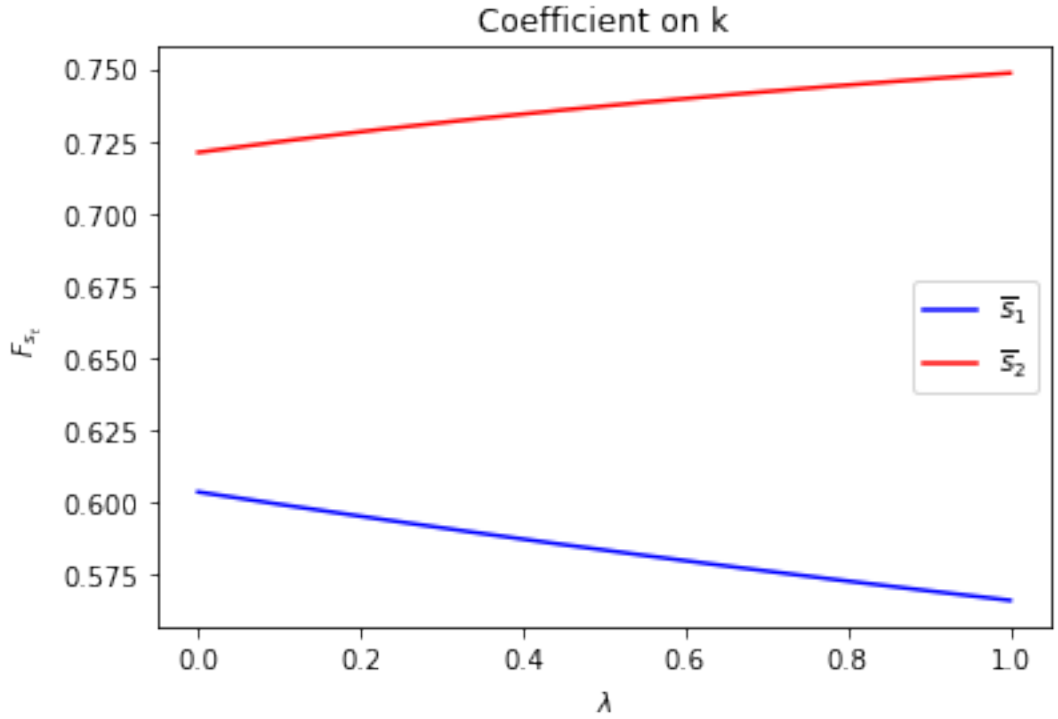
```
λ = 0.2 # low λ
Π2 = np.array([[1-λ, λ],
               [λ, 1-λ]])

ex1_b = qe.LQMarkov(Π2, Qs, Rs, As, Bs, Cs=Cs, Ns=Ns, beta=β)
ex1_b.stationary_values();
ex1_b.Fs
```

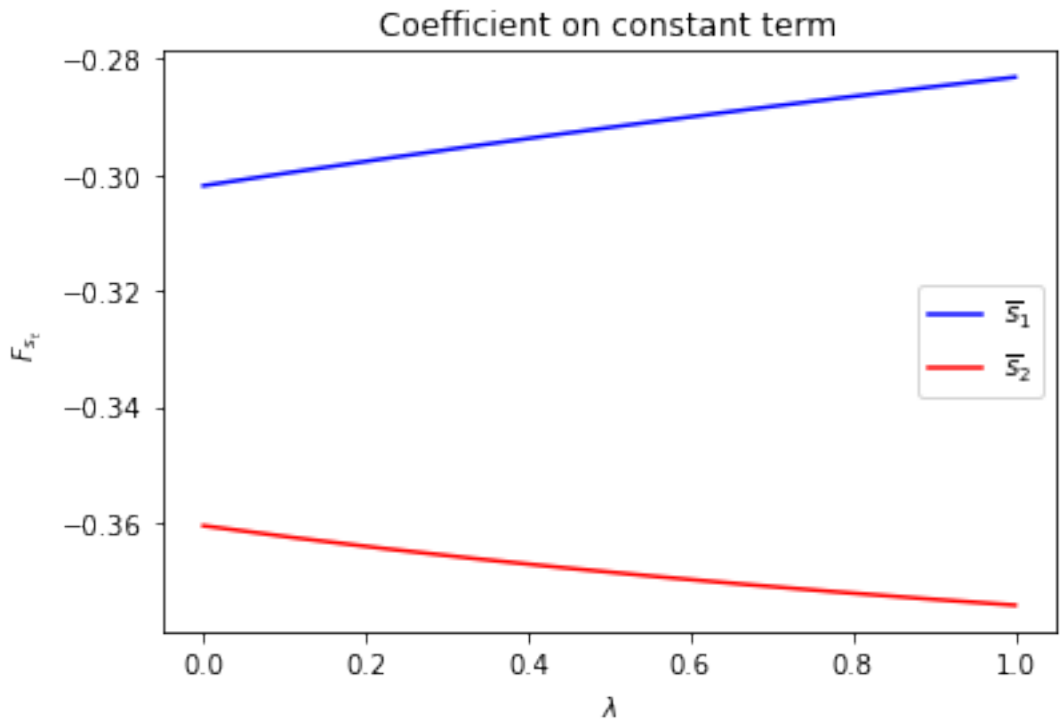
```
array([[[ 0.59533259, -0.2976663 ]],

       [[ 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()
```





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.

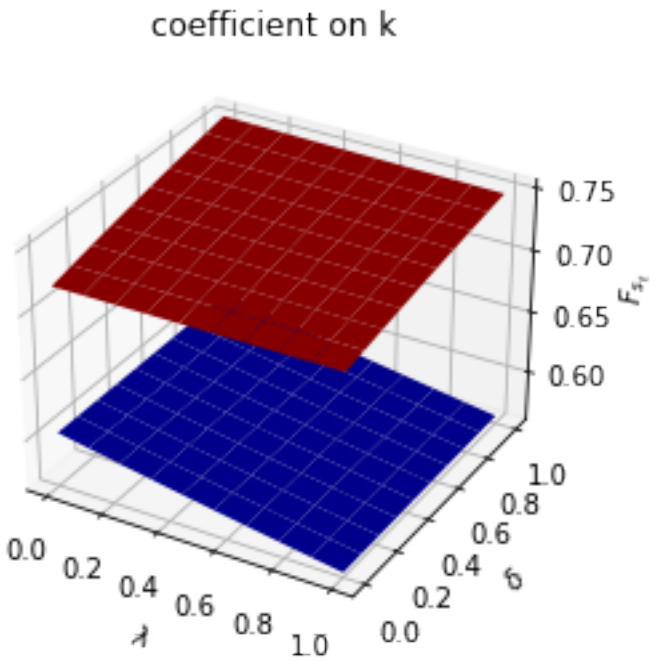
```
λ_vals = np.linspace(0., 1., 10)
δ_vals = np.linspace(0., 1., 10)

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)))
```

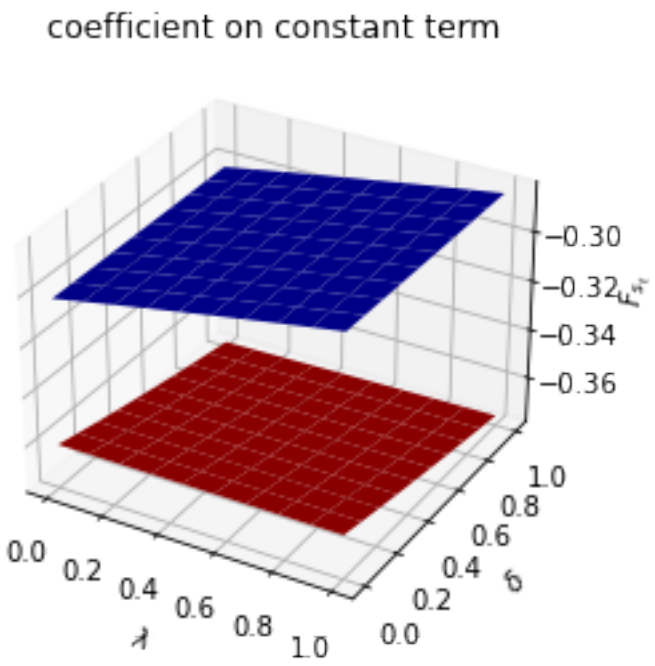
(continues on next page)

(continued from previous page)

```
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")
```

(continues on next page)

(continued from previous page)

```
        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, "$\lambda=0.5$")
        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("$\delta$")
```

(continues on next page)

(continued from previous page)

```
        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 λ.

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 $k^{target}_{s_t} \to k^*_{s_t}$.

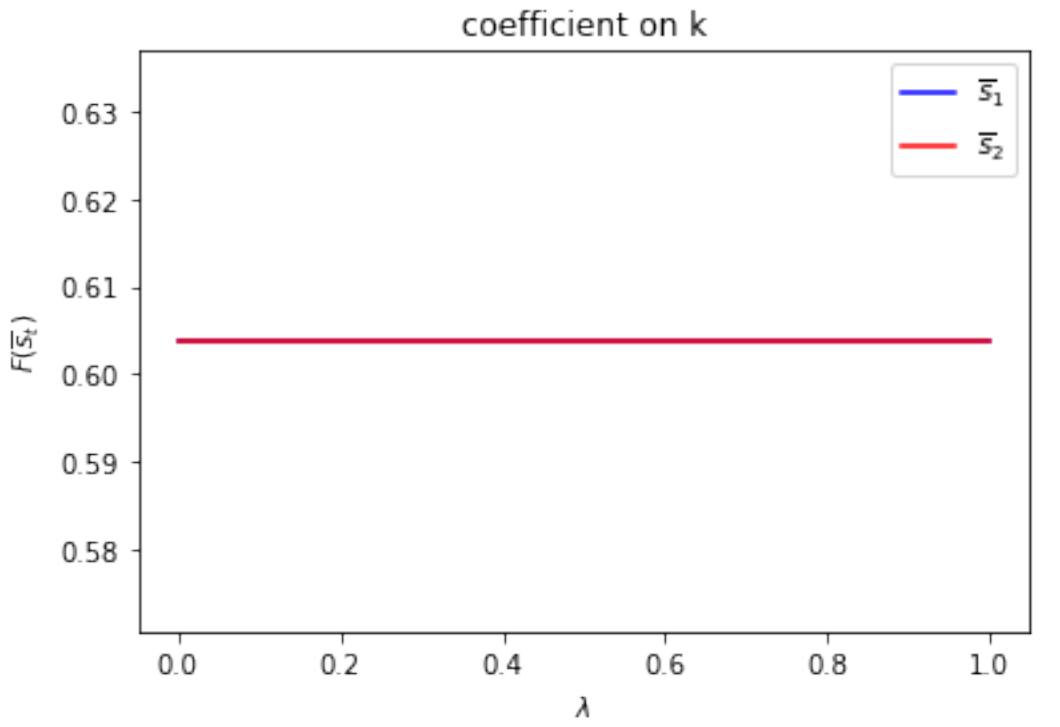
But when $\lambda \to 1$, the Markov transition matrix becomes more nearly periodic, so the optimum decision rules target more at the optimal k level in the other state in order to enjoy higher expected payoff in the next period.

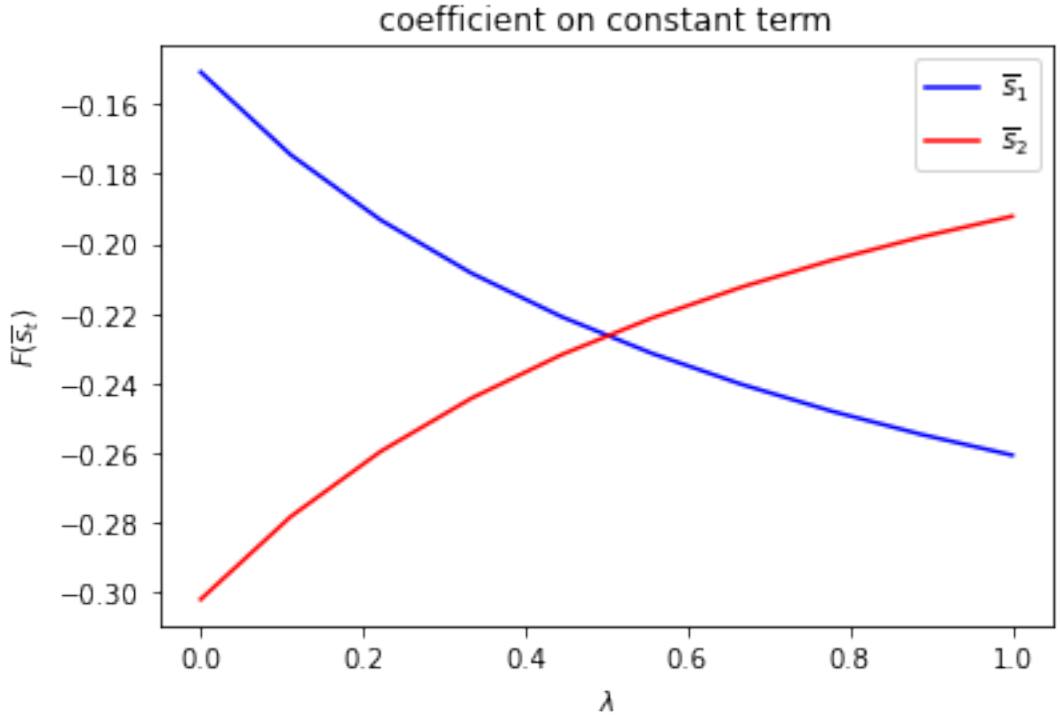
The switch happens at $\lambda = 0.5$ when both states are equally likely to be reached.

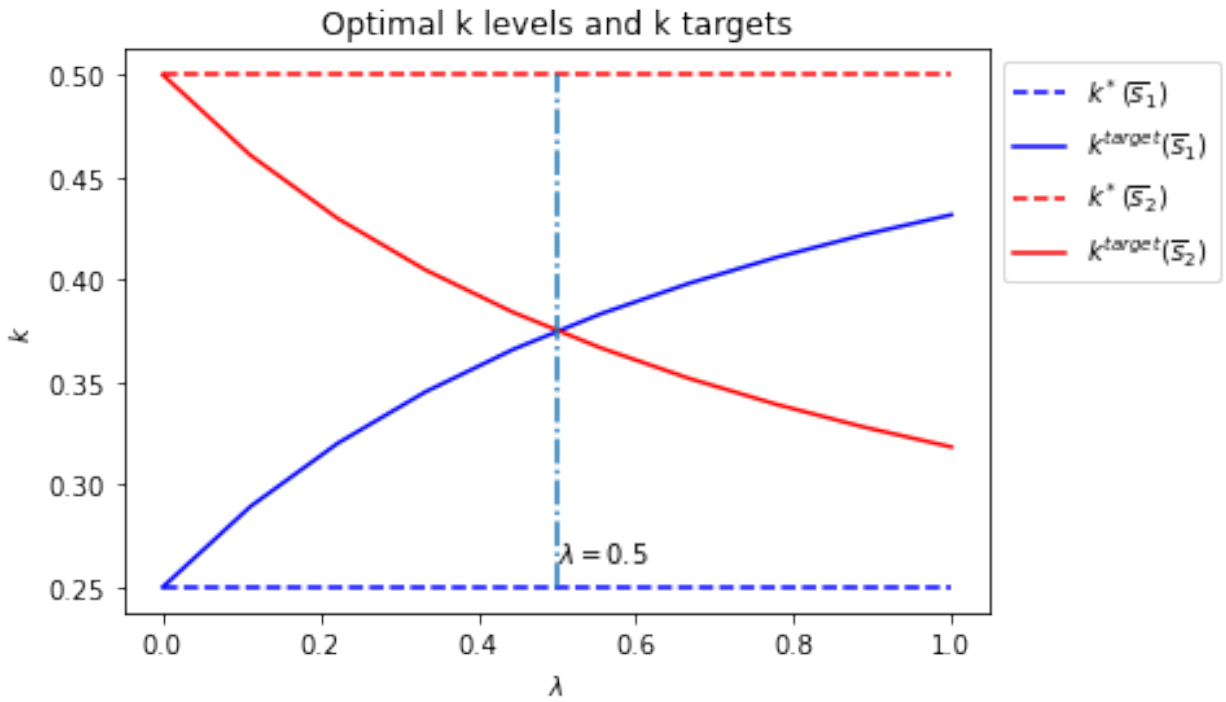
Below we plot an additional figure that shows optimal k levels in the two states Markov jump state and also how the targeted k levels change as λ changes.

```
run(construct_arrays1, {"f1_vals":[0.5, 1.]}, state_vec1)
```

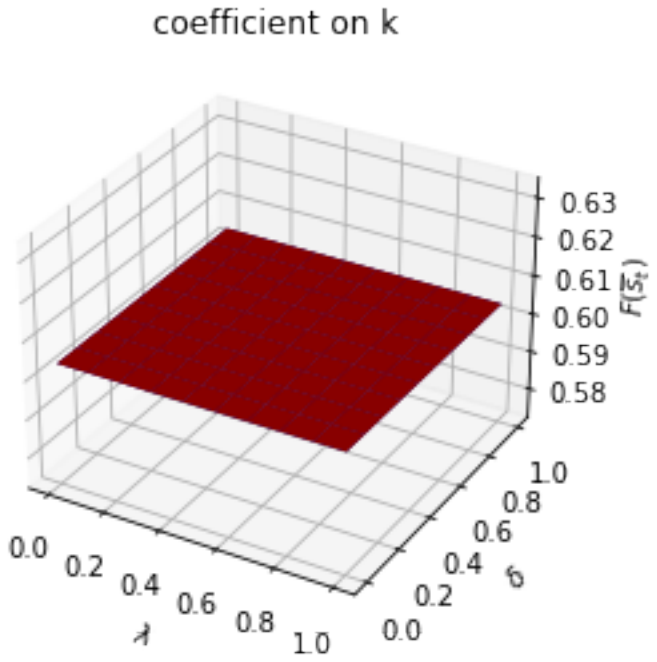
```
symmetric Π case:
```

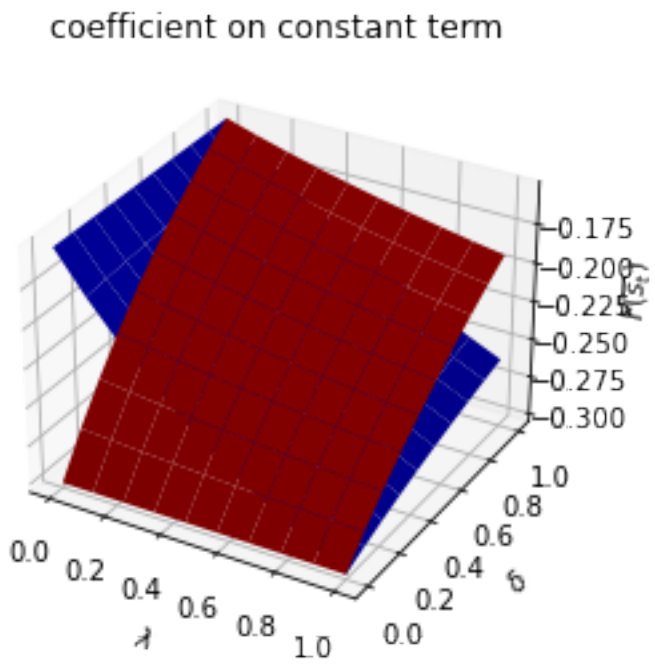

coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
$F(\overline{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ
$\overline{s}_1$
$\overline{s}_2$


coefficient on constant term
−0.16
−0.18
−0.20
−0.22
−0.24
−0.26
−0.28
−0.30
$F(\overline{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ
$\overline{s}_1$
$\overline{s}_2$




```
asymmetric Π case:
```



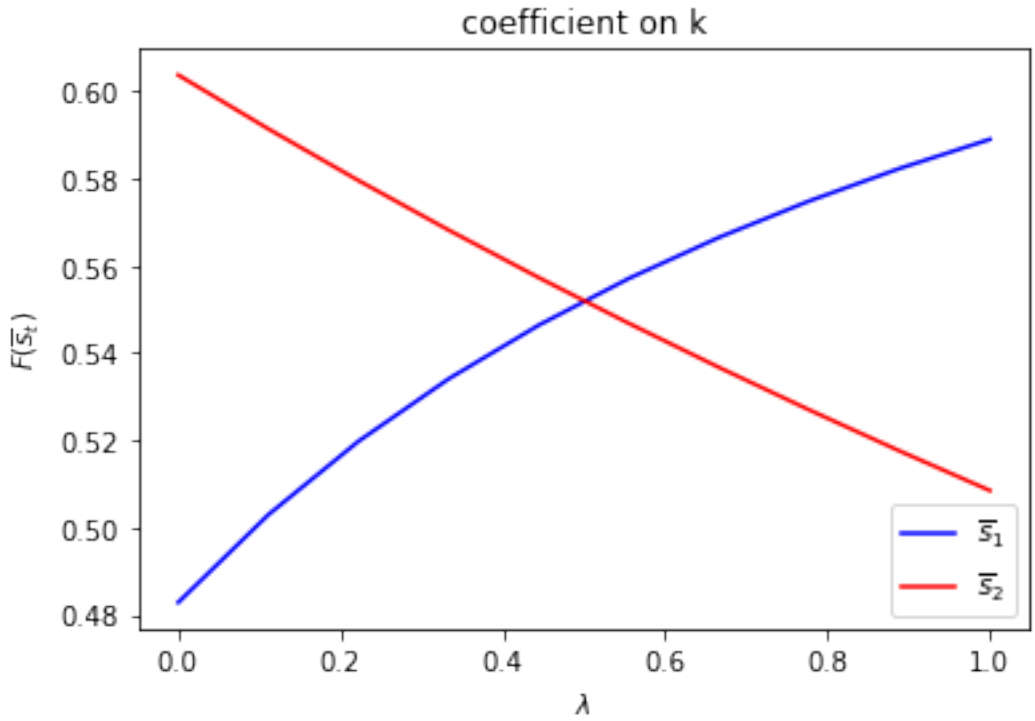


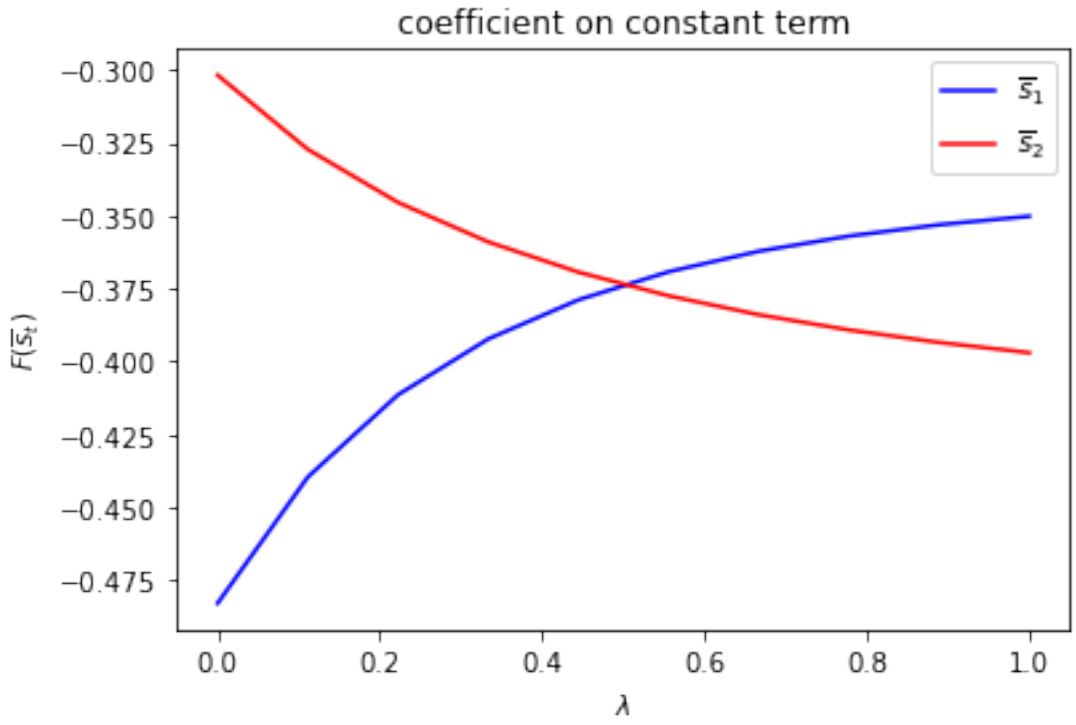
Set f_{1,s_t} and d_{s_t} as constant functions and

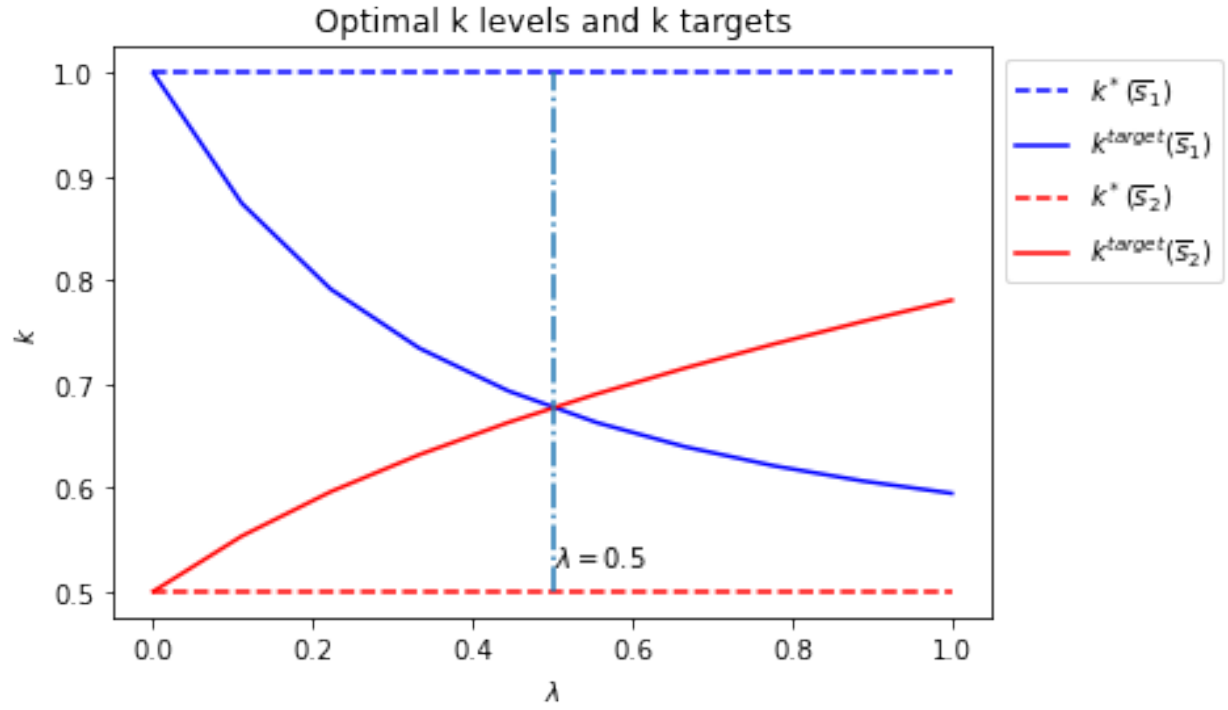
$$f_{2,1} = 0.5, f_{2,2} = 1$$

```
run(construct_arrays1, {"f2_vals":[0.5, 1.]}, state_vec1)
```

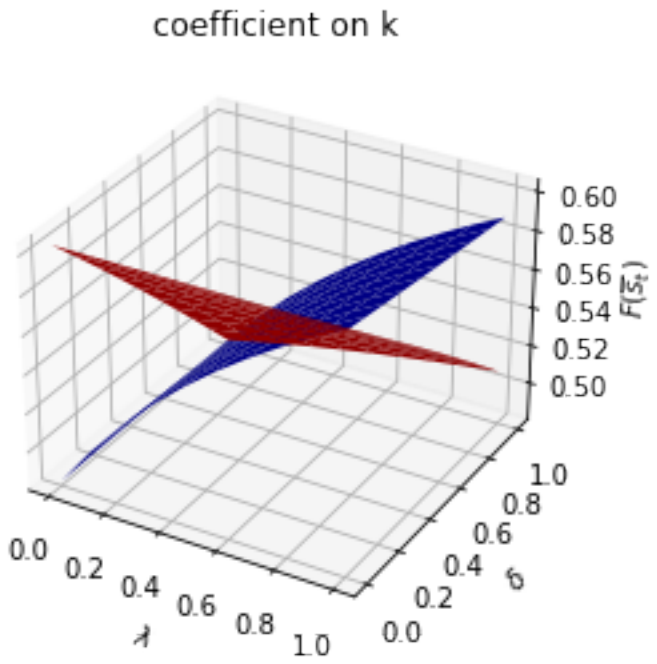
```
symmetric Π case:
```

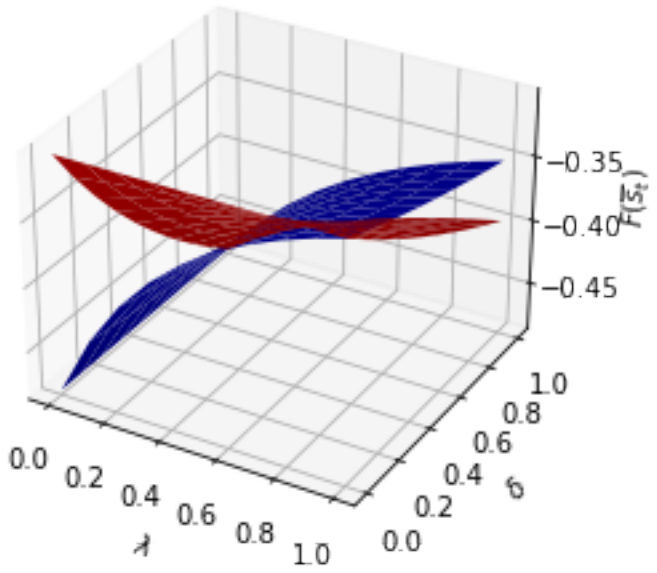






asymmetric Π case:


coefficient on k
0.60
0.58
0.56
0.54
0.52
0.50
$F(\vec{s}_t)$
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
b


coefficient on constant term
−0.35
−0.40
−0.45
$F(\vec{s}_t)$
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
b

9.6 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

$$x_{t+1} = \begin{bmatrix} k_{t+1} \\ 1 \\ w_{t+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \alpha_0(s_t) & \rho(s_t) \end{bmatrix}}_{\equiv A(s_t)} x_t + \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{\equiv B(s_t)} u_t + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \sigma(s_t) \end{bmatrix}}_{\equiv C(s_t)} \epsilon_{t+1}$$

```
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)

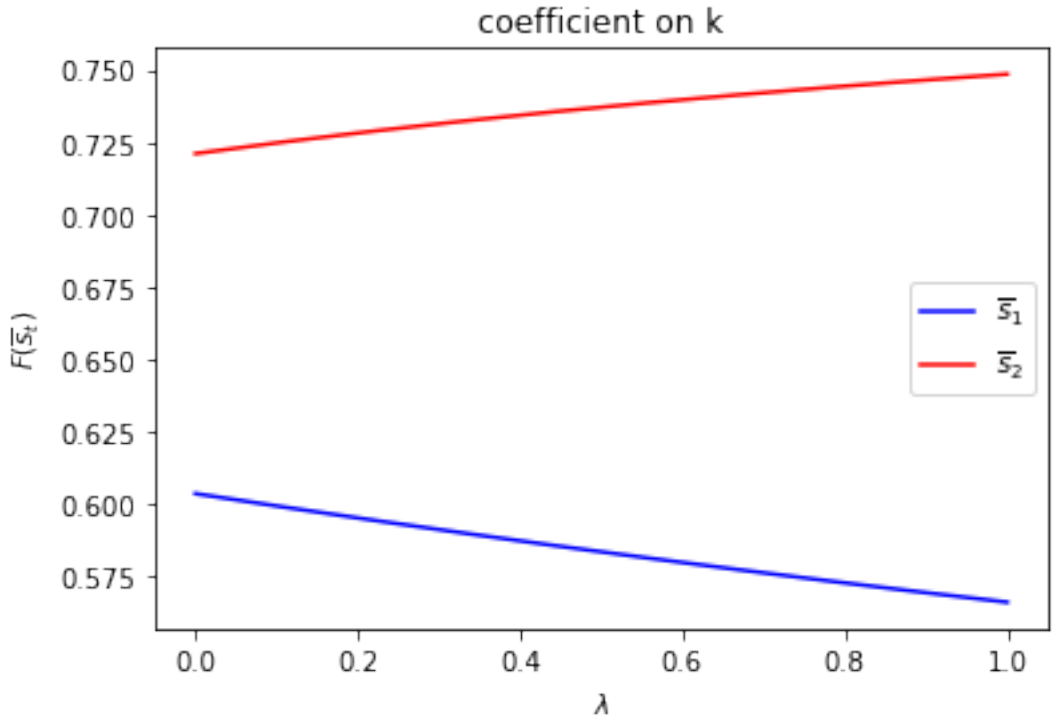
```
        Rs[i, 0, 1] = - f1_vals[i] / 2
        Rs[i, 0, 2] = 1/2
        Rs[i, 2, 0] = 1/2

        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
```

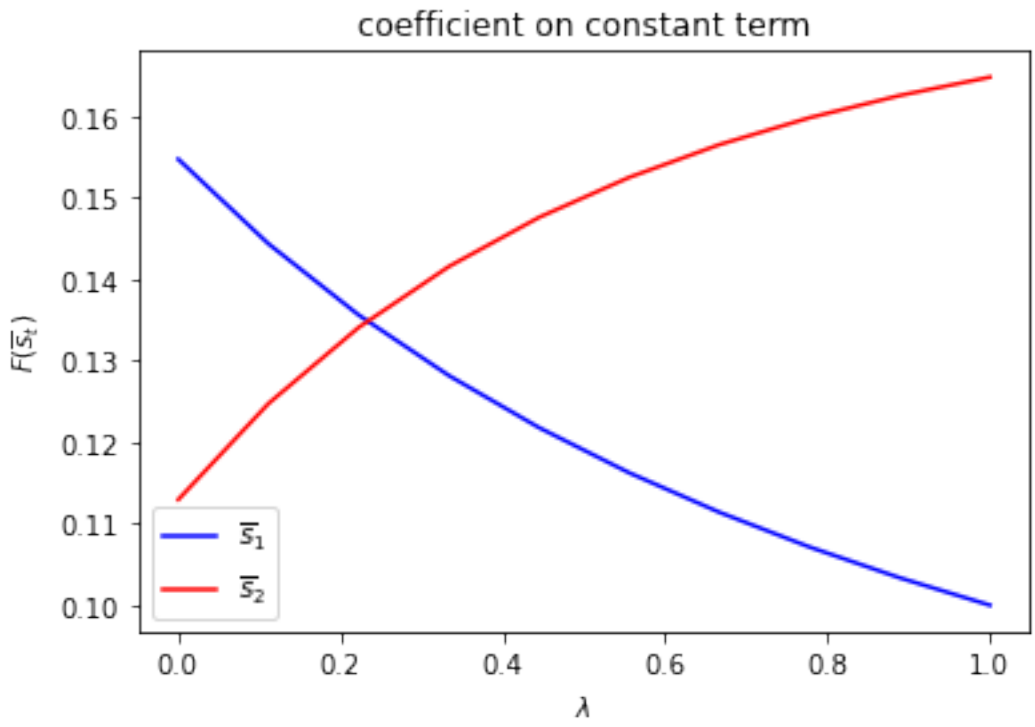
```
state_vec2 = ["k", "constant term", "w"]
```

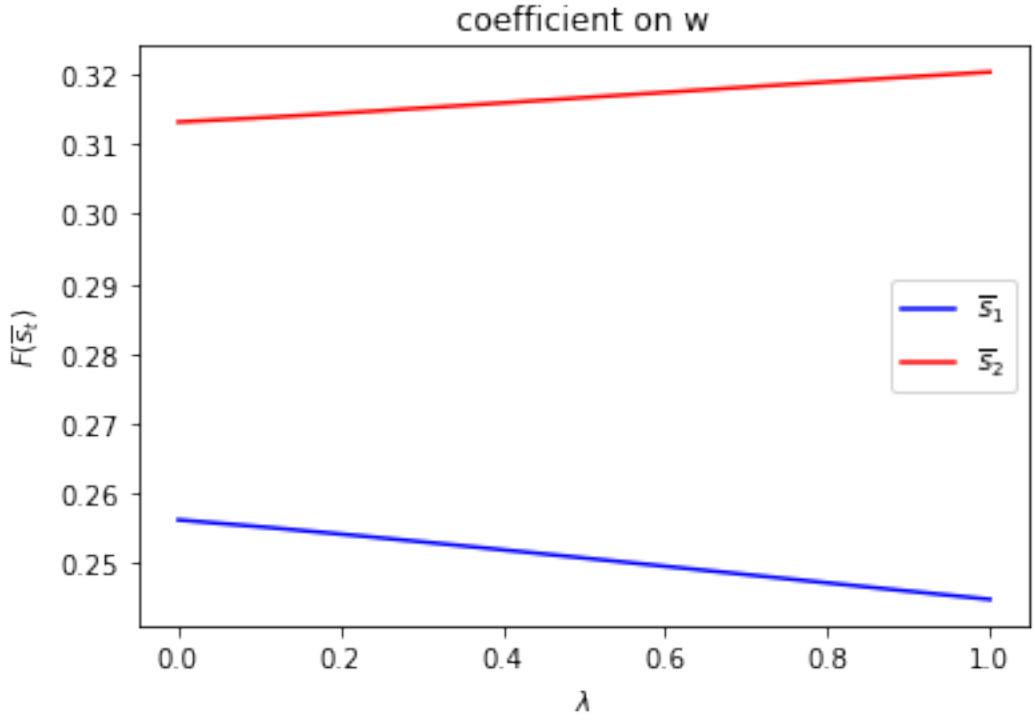
Only d_{s_t} depends on s_t.

```
run(construct_arrays2, {"d_vals":[1., 0.5]}, state_vec2)
```

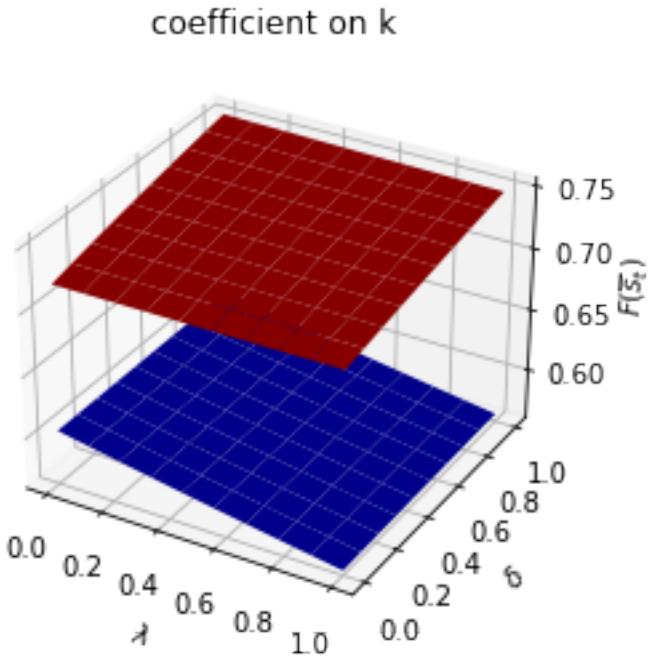
```
symmetric Π case:
```

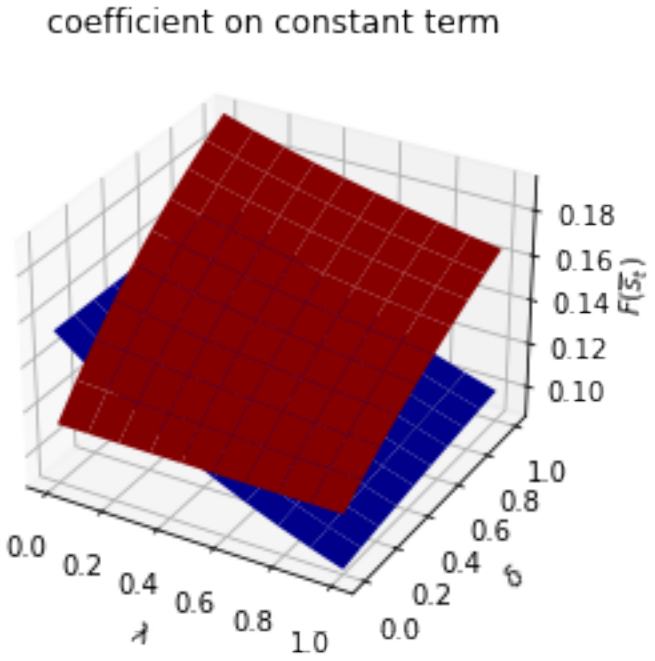


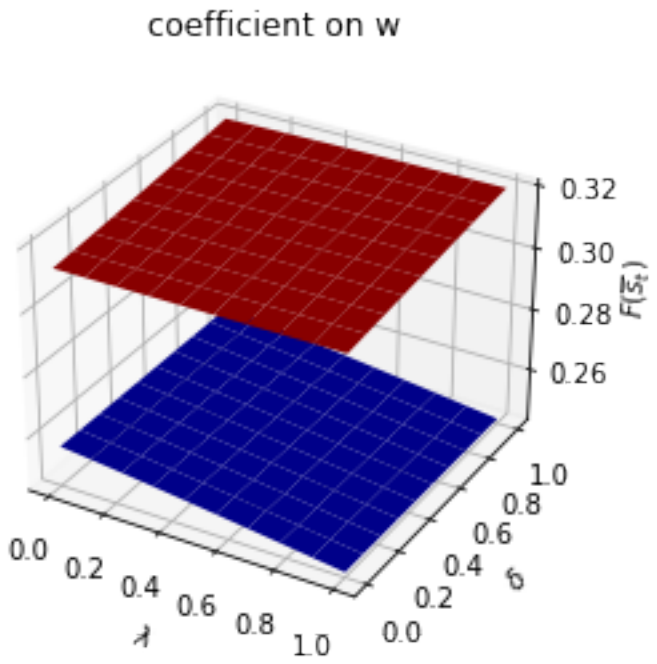





```
asymmetric Π case:
```


coefficient on k
0.75
0.70
0.65
0.60
$F(\vec{s}_t)$
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
b

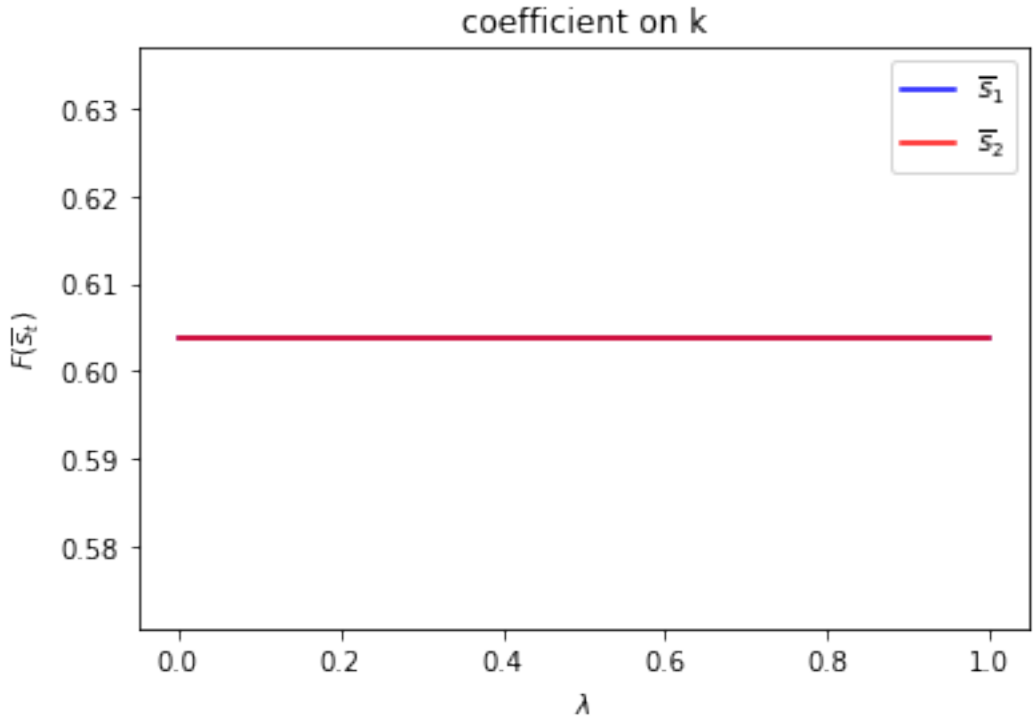

coefficient on constant term
0.18
0.16
0.14
0.12
0.10
$F(\vec{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
b

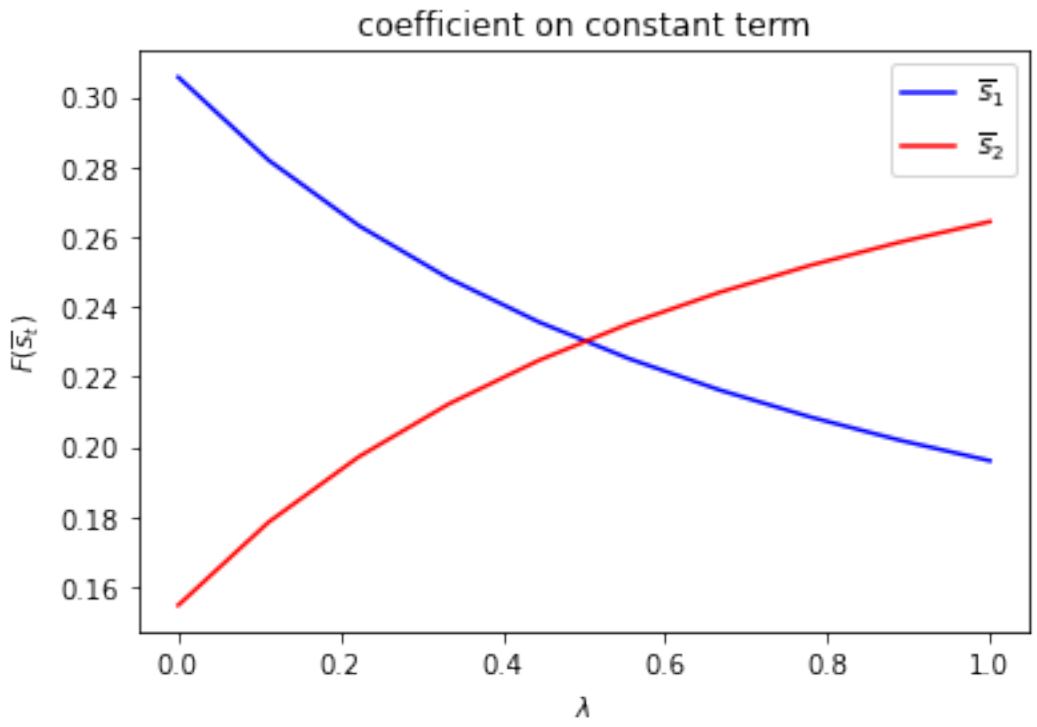


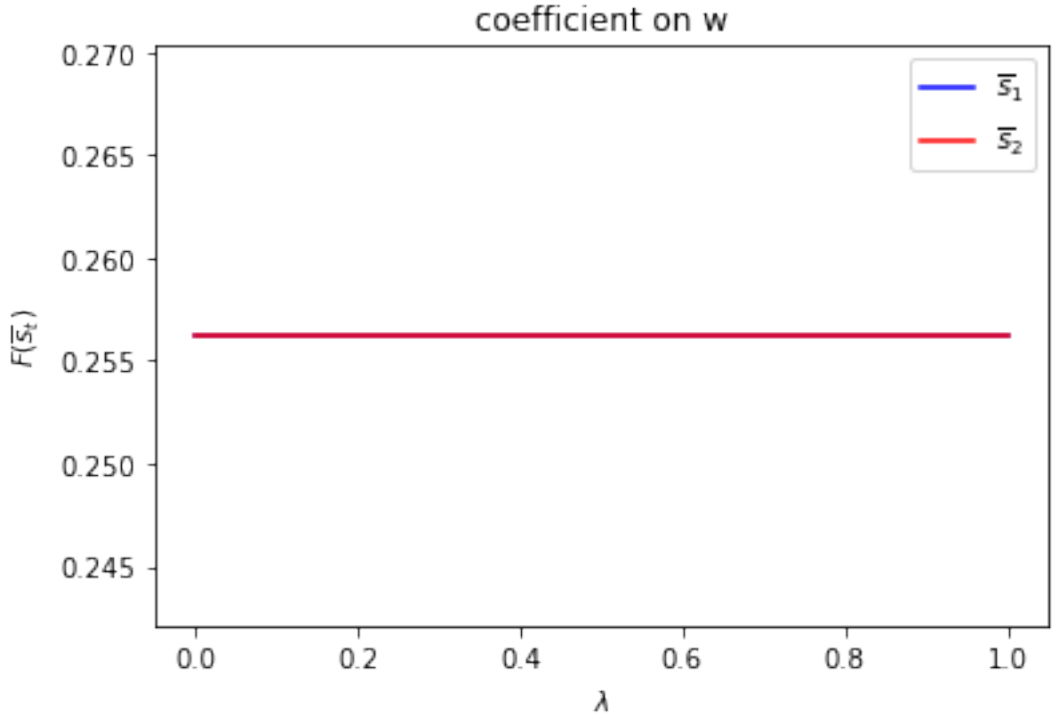
Only f_{1,s_t} depends on s_t.

```
run(construct_arrays2, {"f1_vals":[0.5, 1.]}, state_vec2)
```

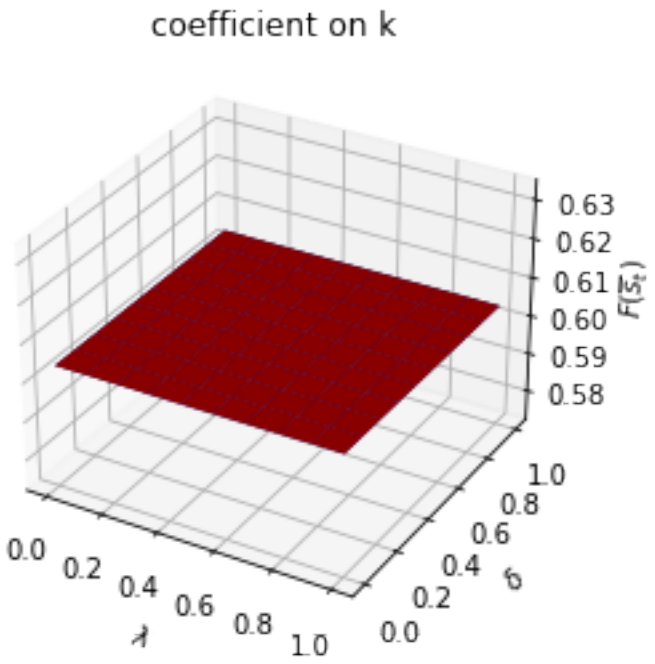
```
symmetric Π case:
```

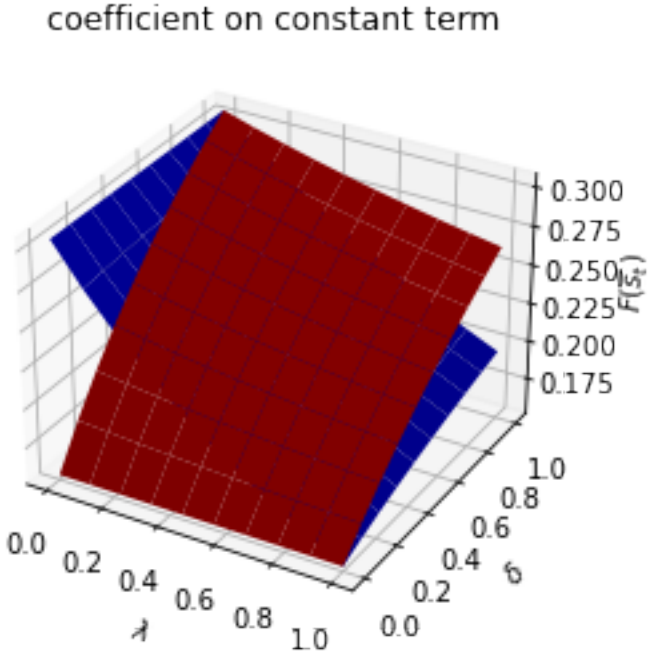


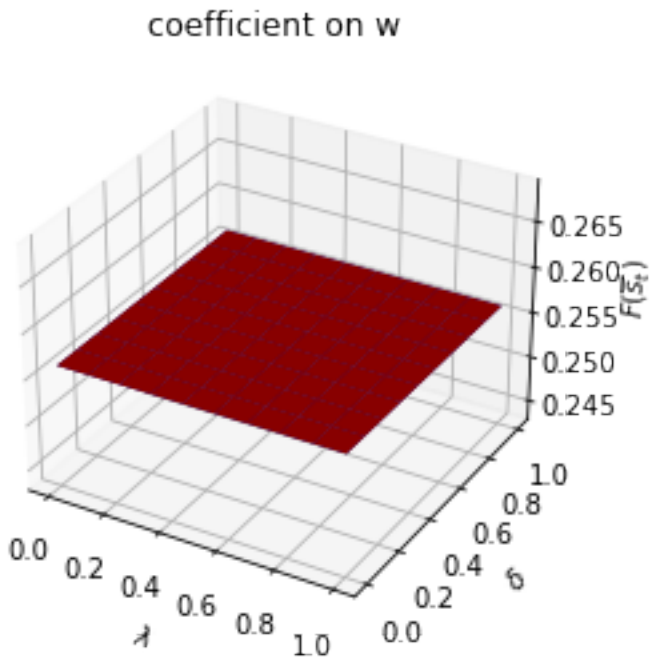





```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
$F(\vec{S}_t)$
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
b

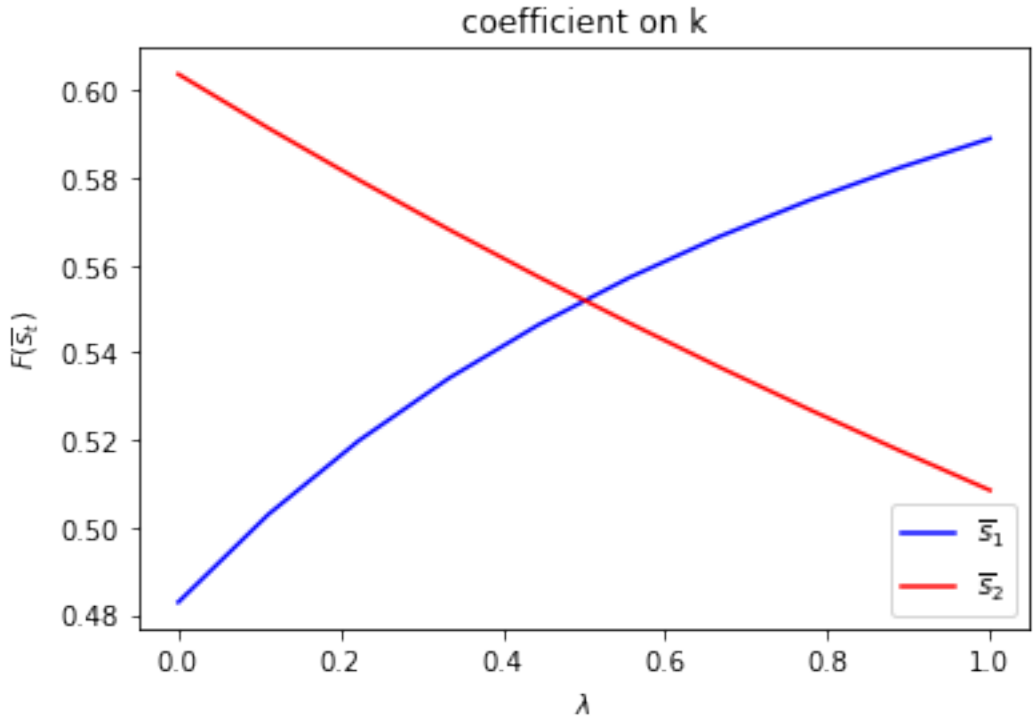

coefficient on constant term
0.300
0.275
0.250
0.225
0.200
0.175
$F(\vec{S}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
b

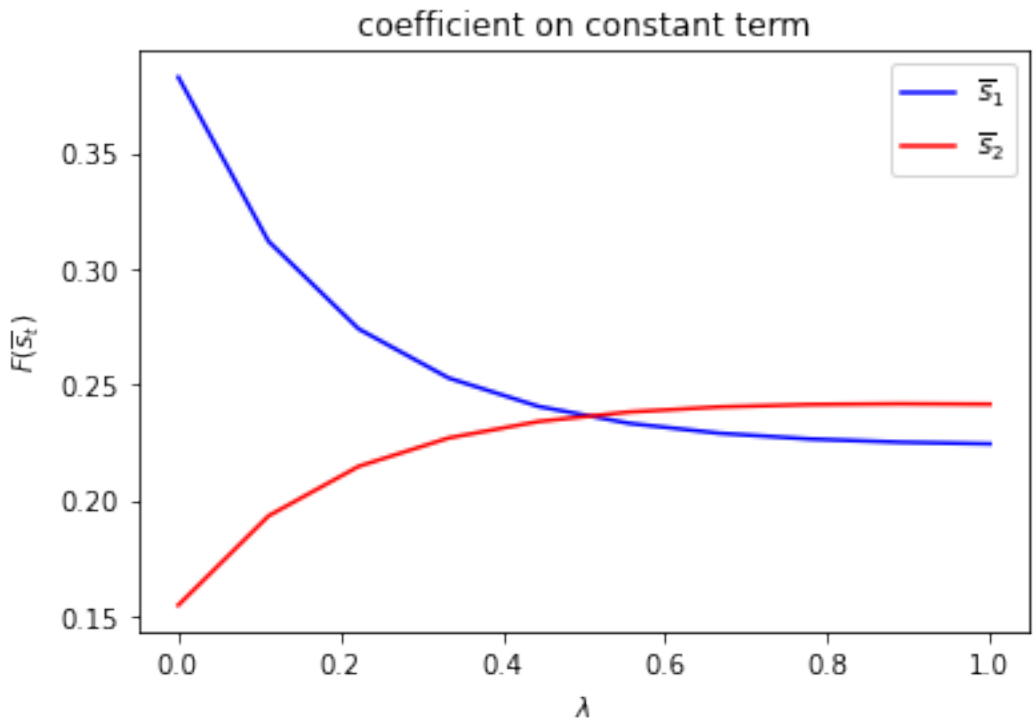


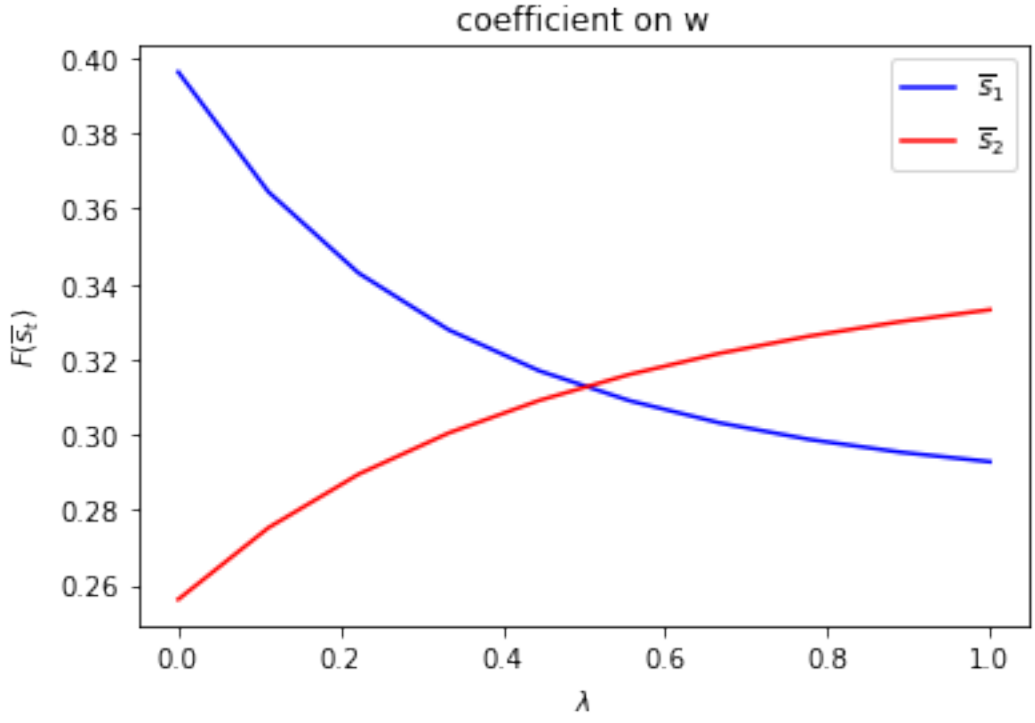
Only f_{2,s_t} depends on s_t.

```
run(construct_arrays2, {"f2_vals":[0.5, 1.]}, state_vec2)
```

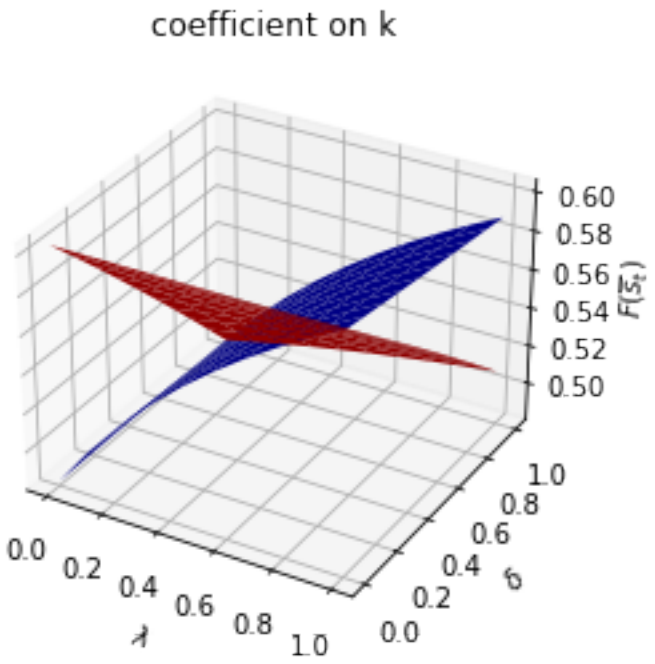
```
symmetric Π case:
```

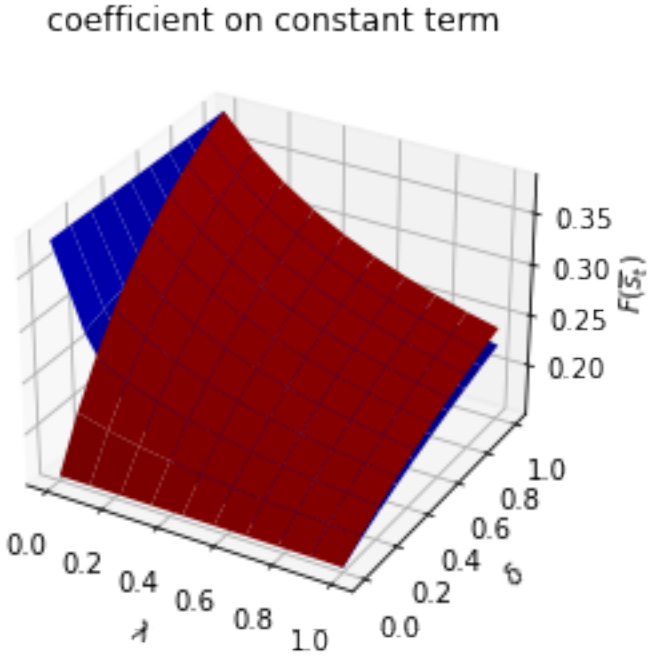


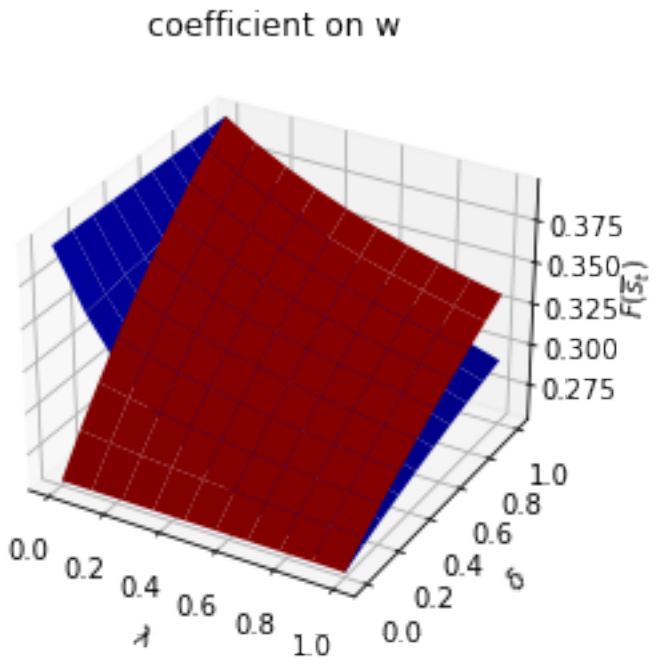





```
asymmetric Π case:
```


coefficient on k
0.60
0.58
0.56
0.54
0.52
0.50
$F(\vec{s}_t)$
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
b

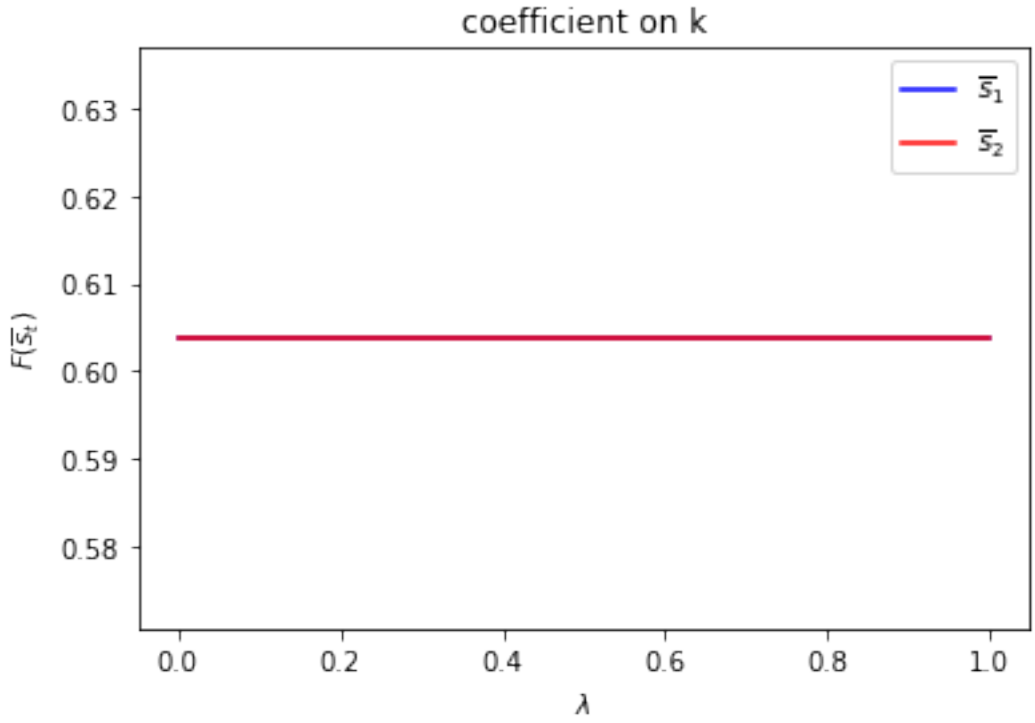

coefficient on constant term
0.35
0.30
0.25
0.20
$F(\vec{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
b

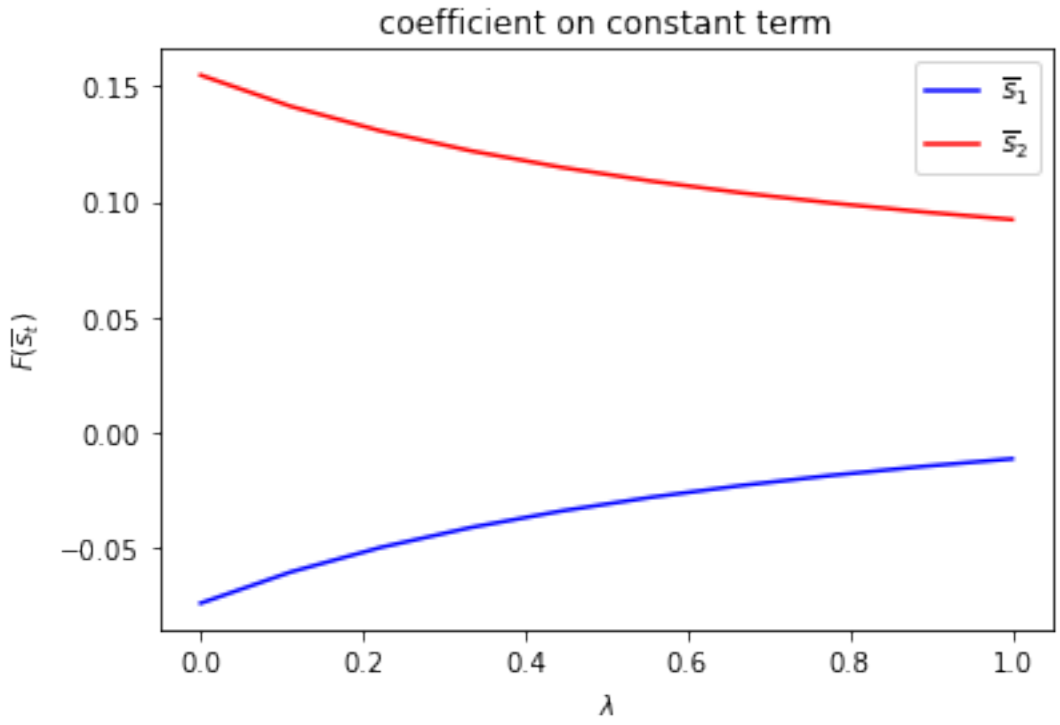


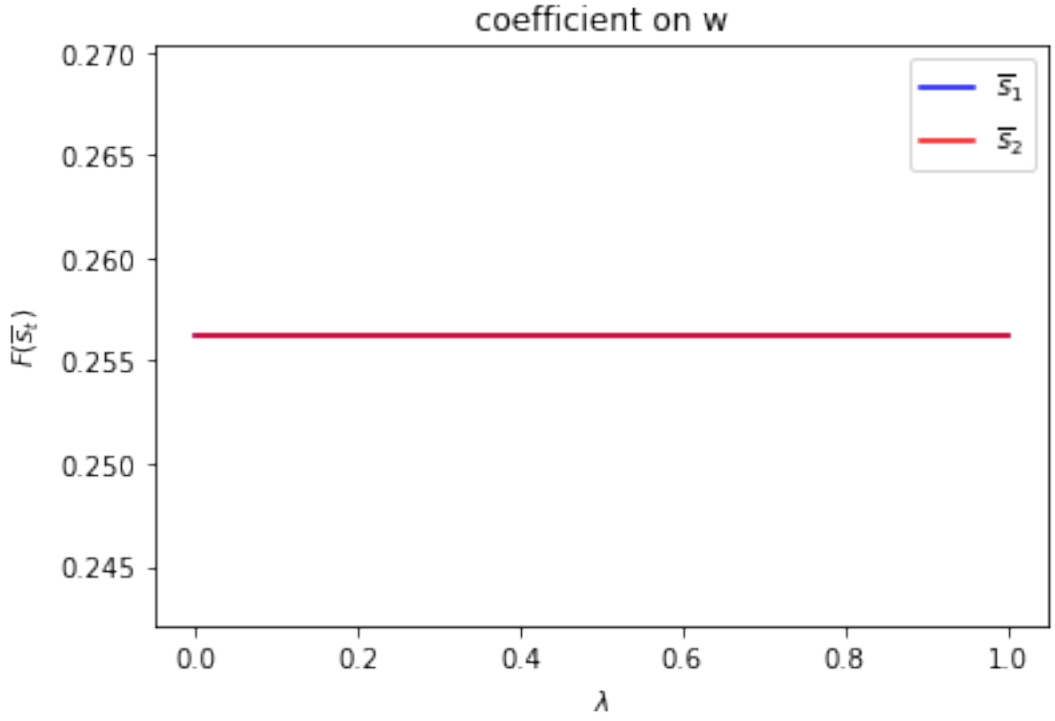
Only $\alpha_0(s_t)$ depends on s_t.

```
run(construct_arrays2, {"α0_vals":[0.5, 1.]}, state_vec2)
```

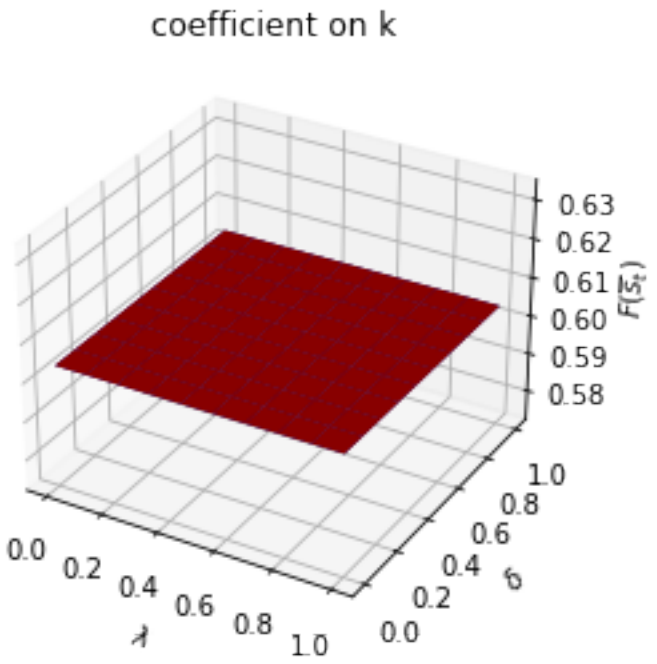
```
symmetric Π case:
```

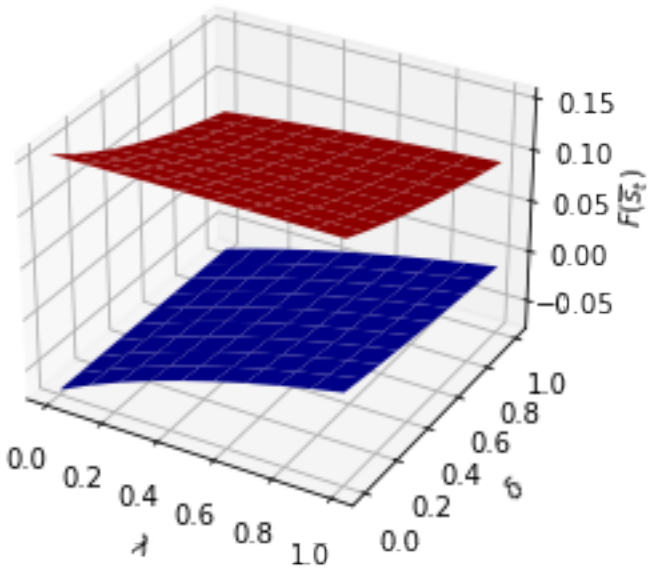


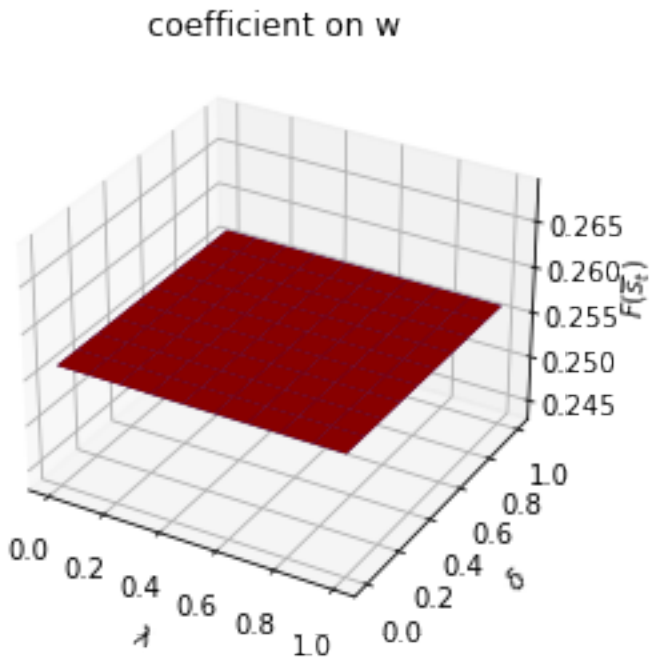





```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
$F(\vec{S}_t)$
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
b

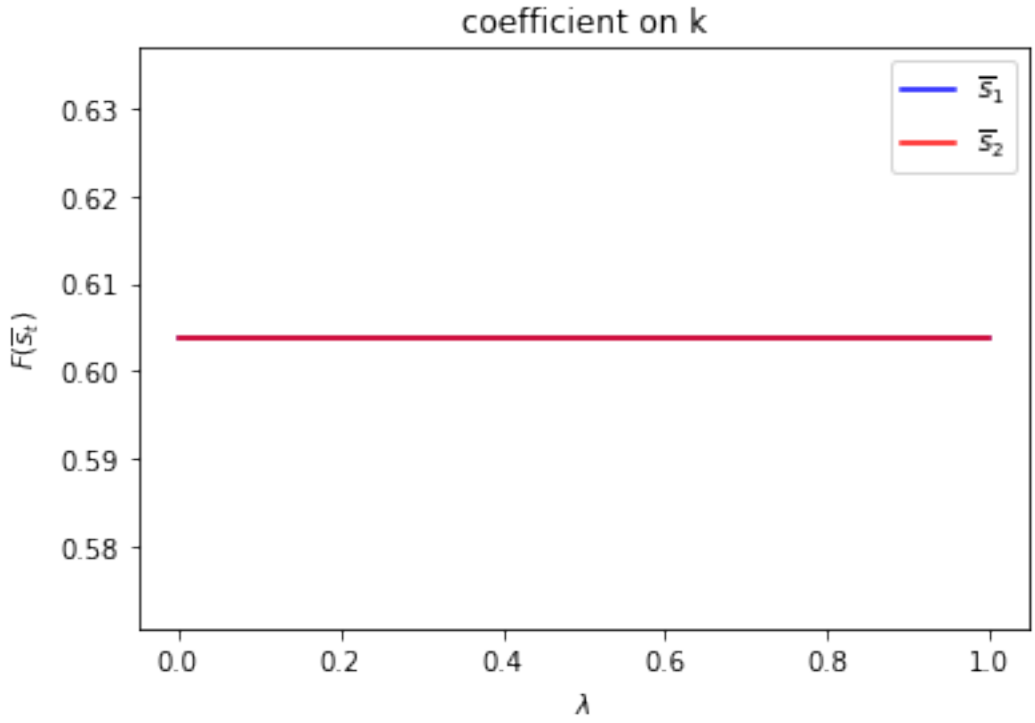

coefficient on constant term
0.15
0.10
0.05
0.00
−0.05
$F(\vec{S}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
b

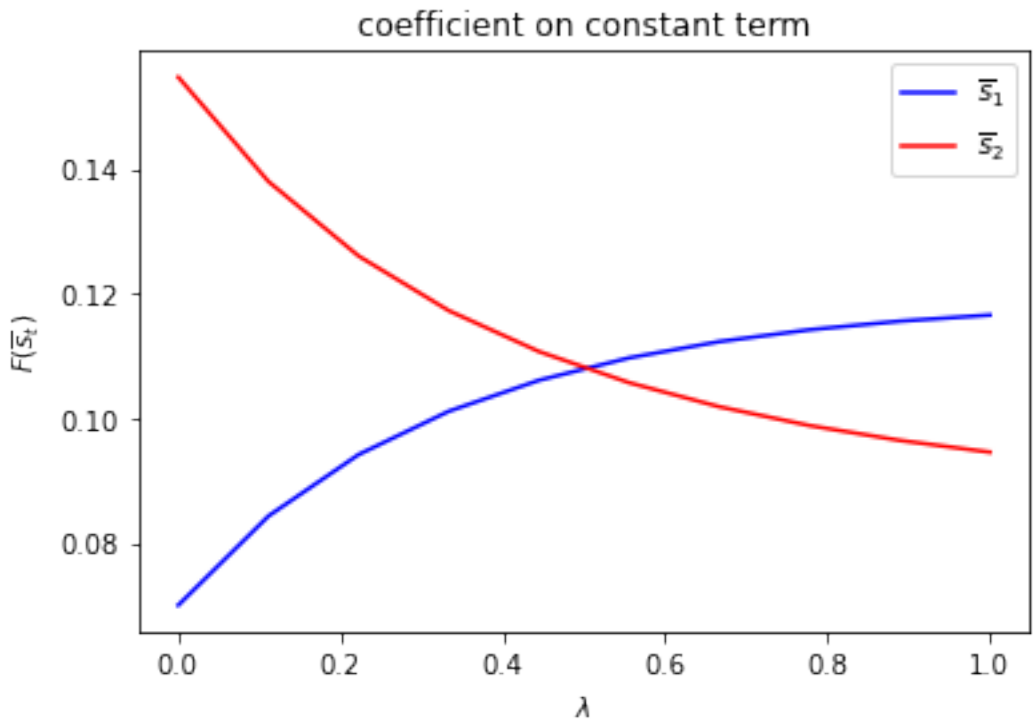


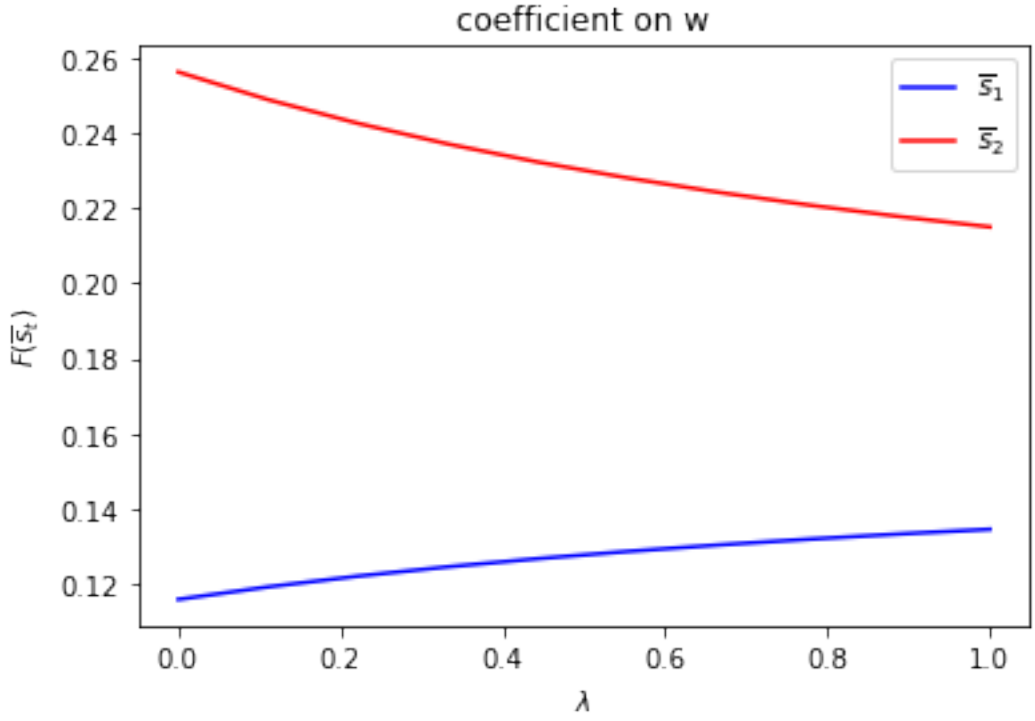
Only ρ_{s_t} depends on s_t.

```
run(construct_arrays2, {"ρ_vals":[0.5, 0.9]}, state_vec2)
```

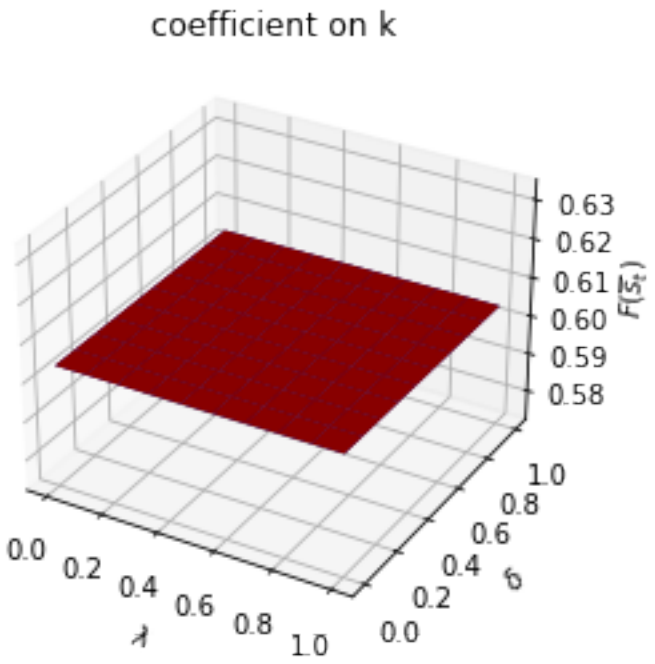
```
symmetric Π case:
```

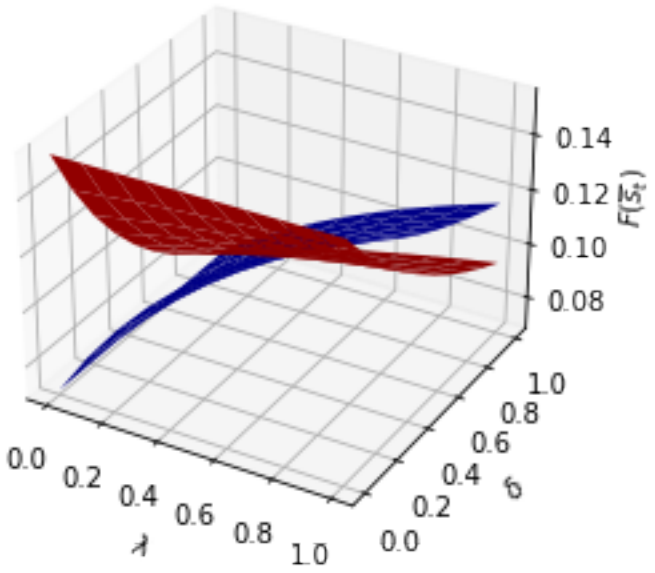


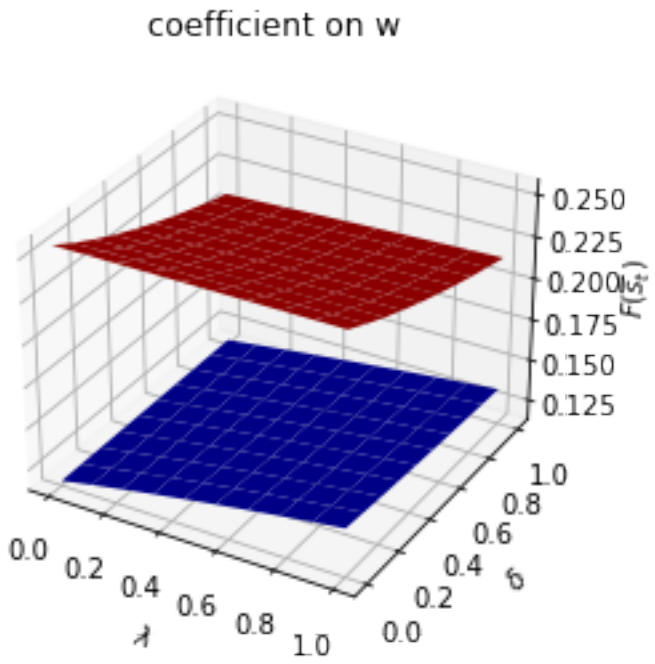





```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
$F(\vec{s}_t)$
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
b

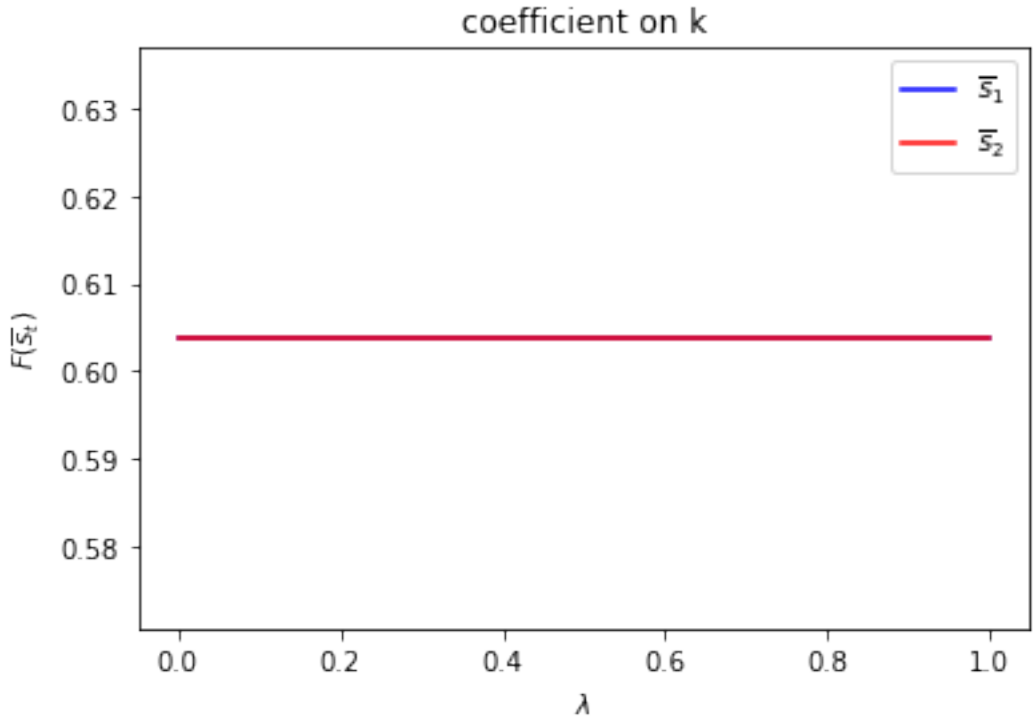

coefficient on constant term
0.14
0.12
0.10
0.08
$F(\vec{s}_t)$
0.0
0.2
0.4
0.6
0.8
1.0
λ
0.0
0.2
0.4
0.6
0.8
1.0
b

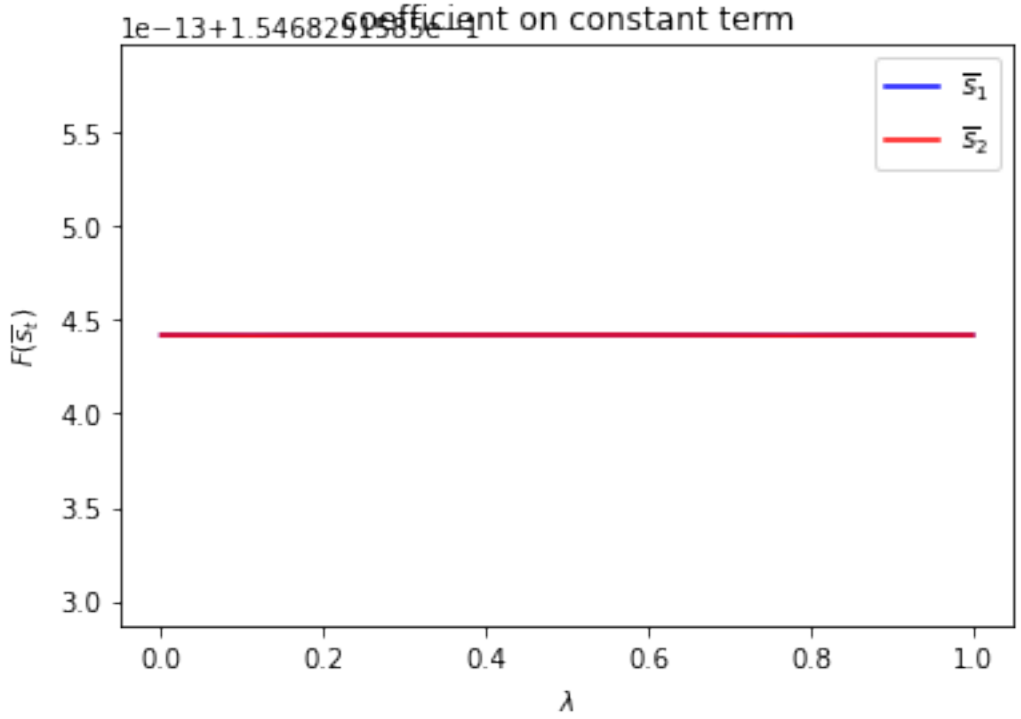


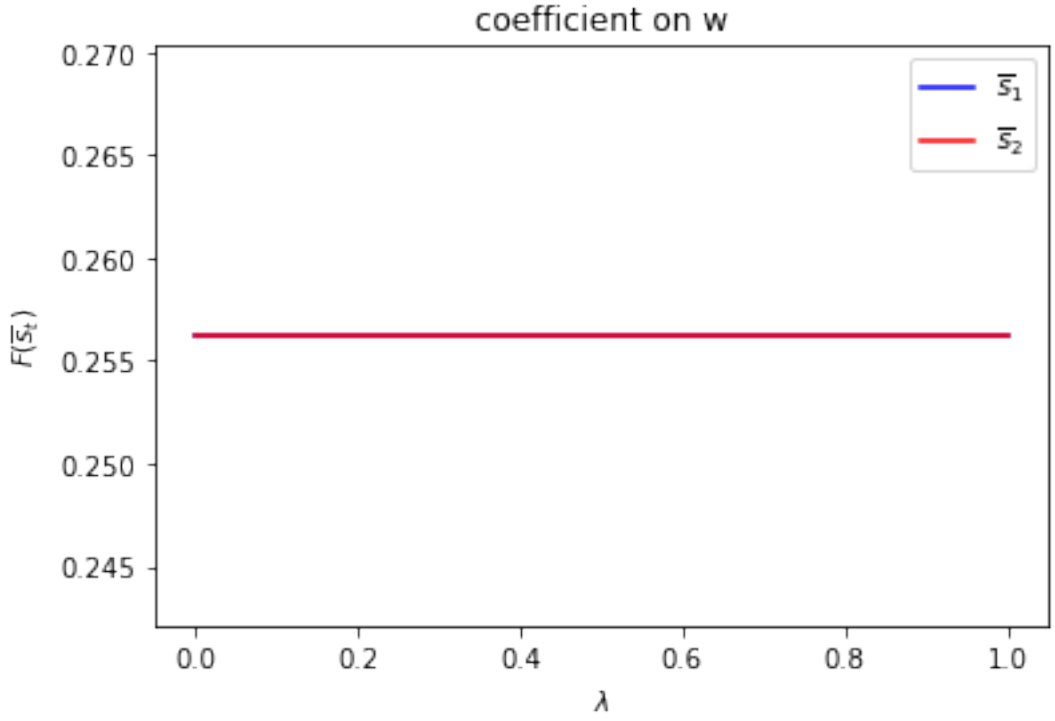
Only σ_{s_t} depends on s_t.

```
run(construct_arrays2, {"σ_vals":[0.5, 1.]}, state_vec2)
```

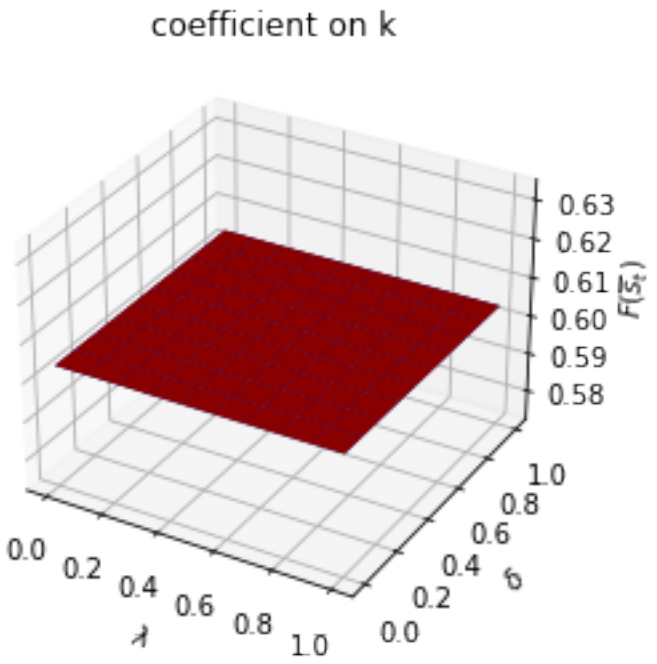
```
symmetric Π case:
```

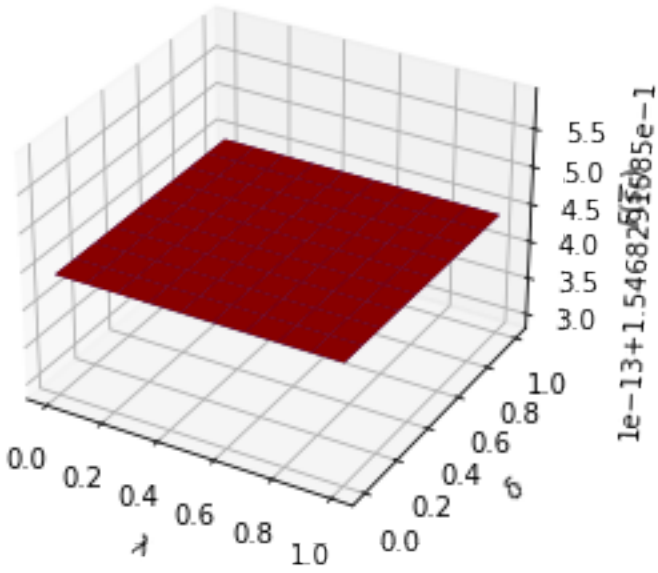


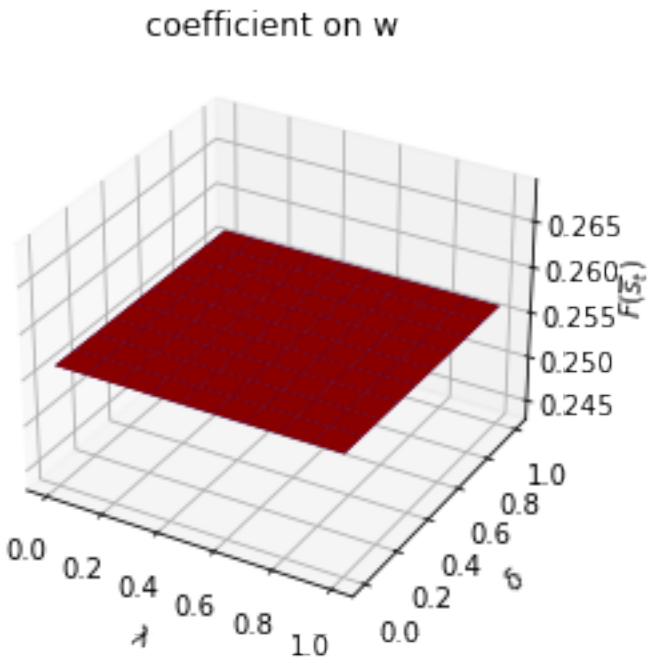





```
asymmetric Π case:
```


coefficient on k
0.63
0.62
0.61
0.60
0.59
0.58
$F(\vec{s}_t)$
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
b


coefficient on constant term
5.5
5.0
4.5
4.0
3.5
3.0
1e−13+1.5468251685e−1
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
b



9.7 More examples

The following lectures describe how Markov jump linear quadratic dynamic programming can be used to extend the [Bar79] 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

TEN

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
```

10.1 Reader's Guide

Let's start with some standard imports:

```
import quantecon as qe
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

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 [Bar79] 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 [Bar99], 2003 [BM03]).

Barro's original 1979 [Bar79] 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 [Bar79] 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 [Bar79] model along lines he suggested in Barro (1999 [Bar99], 2003 [BM03]).

Barro (1979) [Bar79] 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) [Bar99] and Barro (2003) [BM03], our generalizations of Barro's (1979) [Bar79] 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.

10.2 Public Finance Questions

Barro's 1979 [Bar79] 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 [Bar99] and 2003 [BM03] 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.

10.3 Barro (1979) Model

We begin by solving a version of the Barro (1979) [Bar79] 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) [Bar79], 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

$$(S - MF)(A - BF) = (S - MF),$$

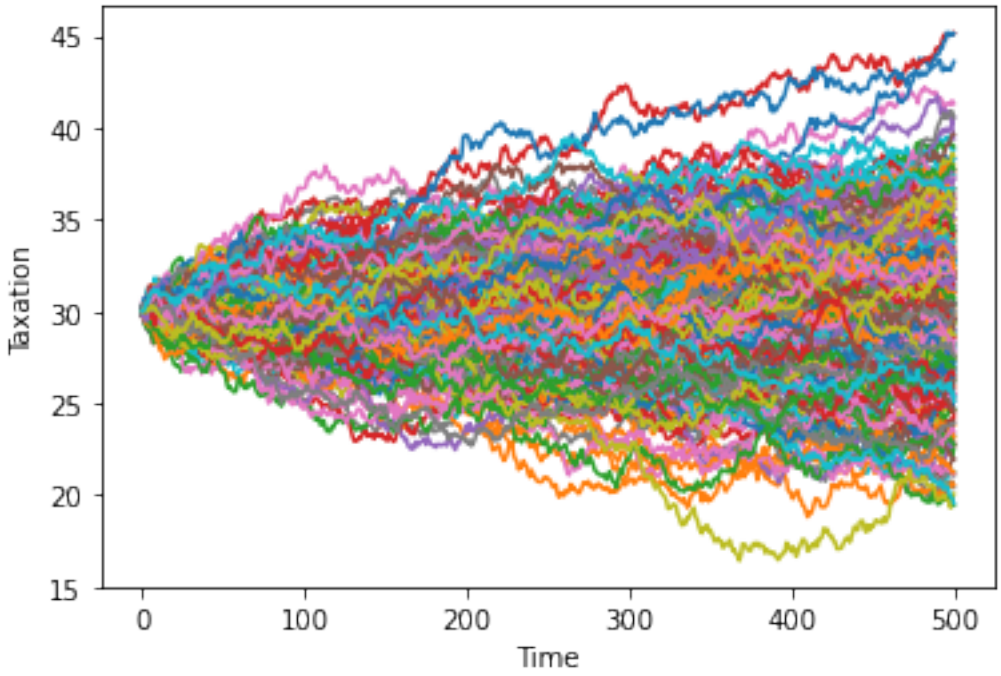
which holds in this case:

```
S - M @ F, (S - M @ F) @ (A - B @ F)
```

```
(array([[ 0.05000002, 19.79166502,  0.2083334 ]]),
 array([[ 0.05000002, 19.79166504,  0.2083334 ]]))
```

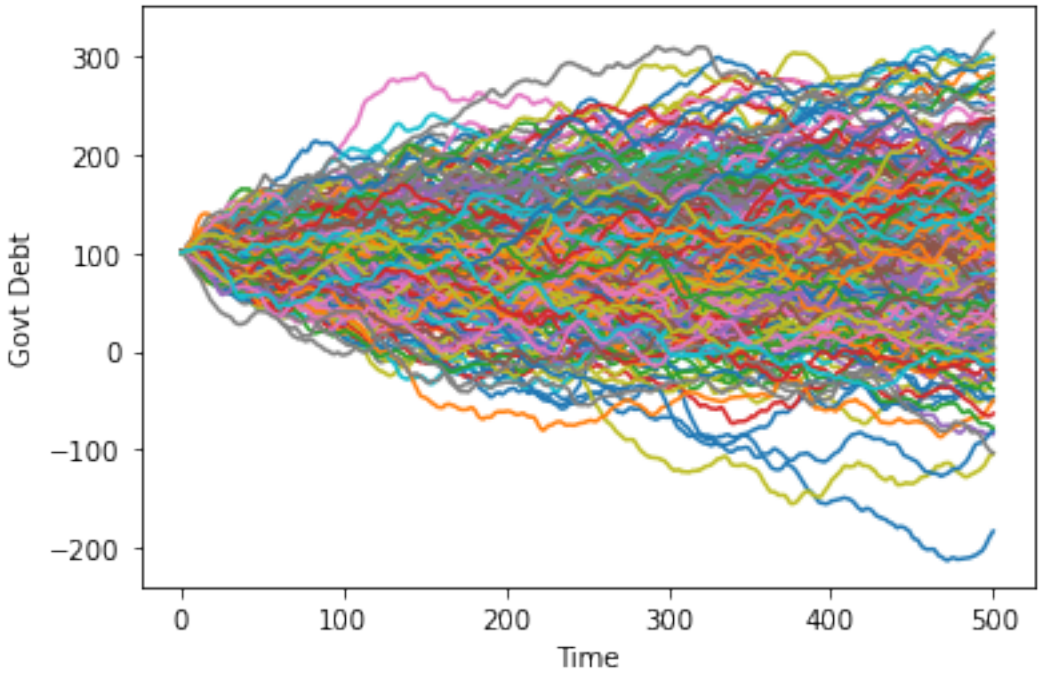
This explains the fanning out of the conditional empirical distribution of taxation across time, computing by simulation the Barro model a large number of times:

```
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()
```



10.4 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.

10.5 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.

```
# 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:

```
lqm.Fs[0]
```

```
array([[-0.98437712, 19.20516427, -0.8314215 ]])
```

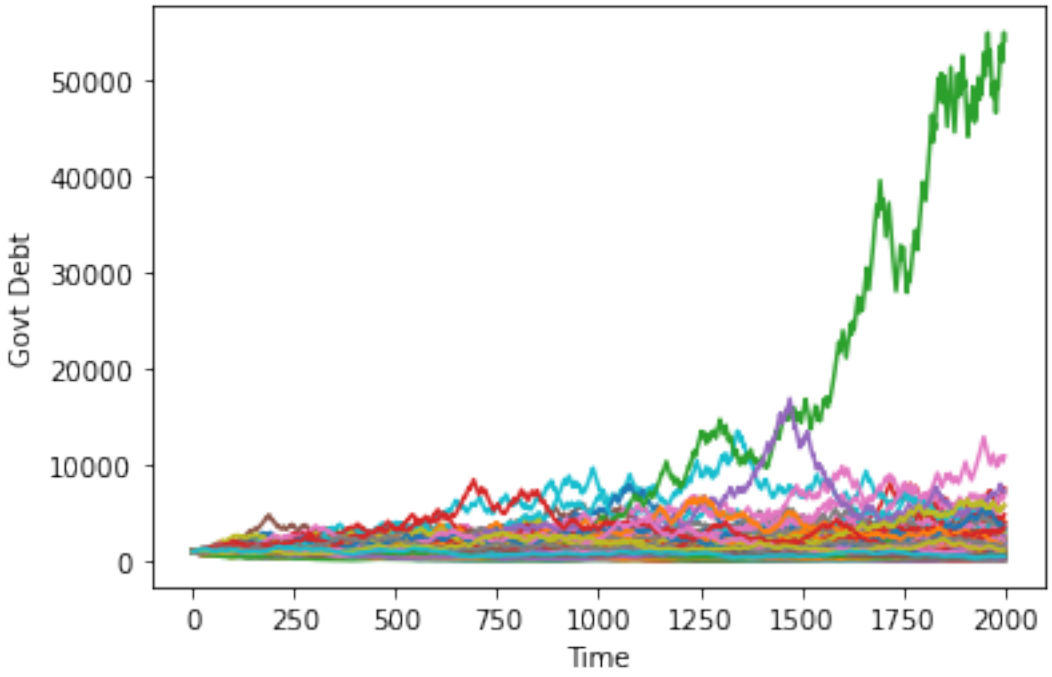
```
lqm.Fs[1]
```

```
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()
```


50000
40000
30000
20000
10000
0
Govt Debt
0
250
500
750
1000
1250
1500
1750
2000
Time

CHAPTER

ELEVEN

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
```

11.1 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 [Bar99], 2003 [BM03]) for extending his classic 1979 model of tax smoothing.

Barro's 1979 [Bar79] 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 [Bar79] model looks a lot like a consumption-smoothing model.

Our generalizations of his 1979 [Bar79] model will also look like souped-up consumption-smoothing models.

Wanting tractability induced Barro in 1979 [Bar79] 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
%matplotlib inline
```

11.2 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.

11.3 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) [Bar79], 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.

11.4 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:

```
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.

11.5 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:

$$A_{22} = \begin{bmatrix} 1 & 0 \\ \bar{G} & \rho \end{bmatrix} \ , \ C_2 = \begin{bmatrix} 0 \\ \sigma \end{bmatrix} \ , \ U_g = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

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:

$$p^1_{t,t+1} = \beta \ , \ p^1_{t,t+2} = \beta^2 - 0.02$$

and in state 2, prices are:

$$p^2_{t,t+1} = \beta \ , \ p^2_{t,t+2} = \beta^2 + 0.02$$

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:

$$\Pi = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$$

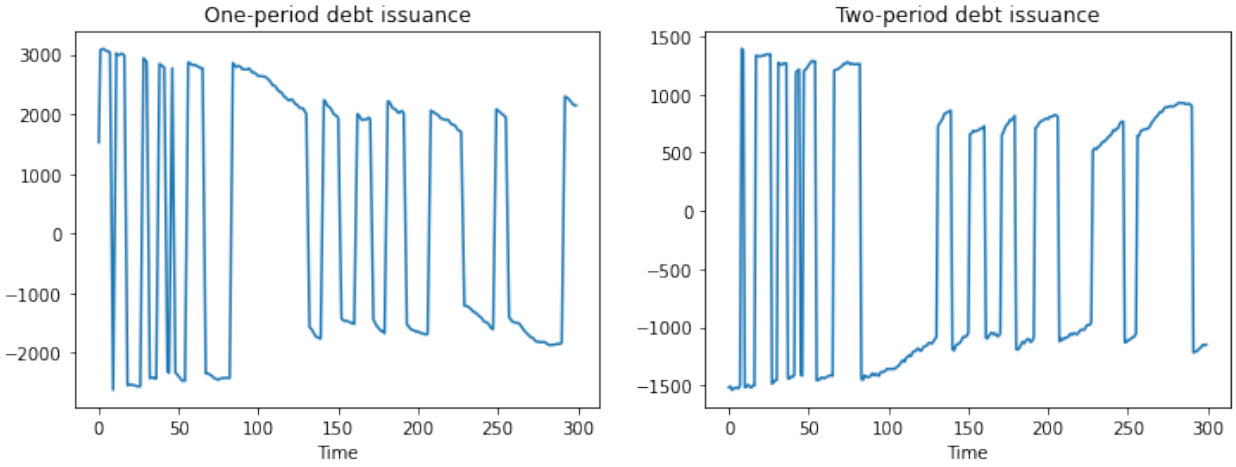
Thus, each Markov state is persistent, and there is an equal chance of moving from one to the other.

```
# 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 $c_1 = 0.01$.

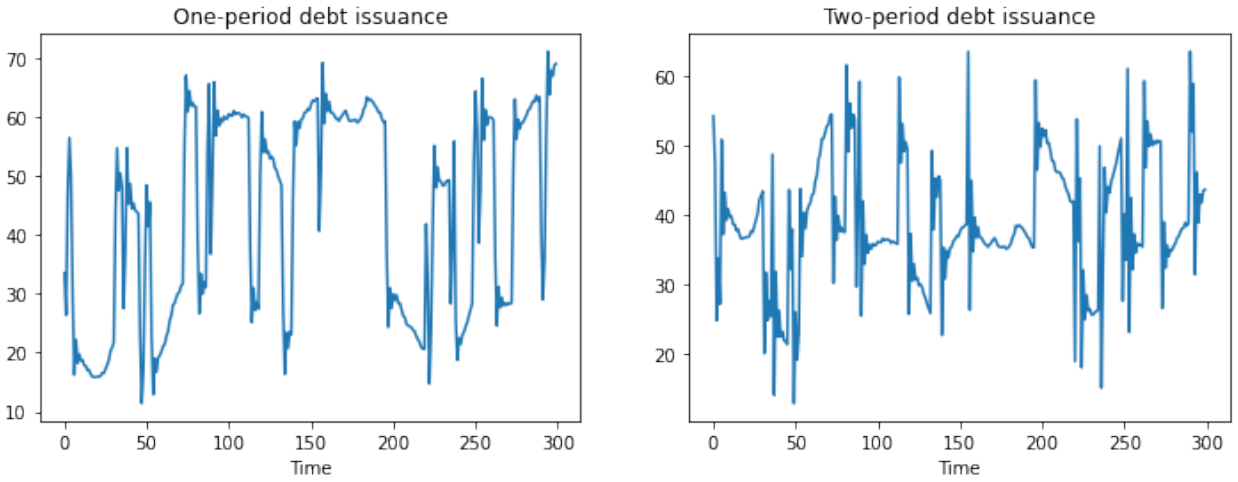
This penalty is enough to ensure that the government issues positive quantities of both one and two-period debt:

```
# 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()
```



11.6 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 $S_t = \begin{bmatrix}(\tilde{S_x} + p_t\text{'}S_s\text{'}S_x) & Ug\end{bmatrix}$.

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

$$R_t = S_t' S_t, \qquad Q_t = p_t p_t', \qquad W_t = -p_t S_t$$

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

$$A_t = \begin{bmatrix} 0 & 0 \\ 0 & A_{22,t} \end{bmatrix}, \qquad B = \begin{bmatrix} I \\ 0 \end{bmatrix}, \qquad C = \begin{bmatrix} 0 \\ C_{2,t} \end{bmatrix}$$

This completes the mapping into a Markov jump LQ problem.

11.7 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:

```
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
```

11.7.1 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:

$$p^1_{t,t+1} = 0.9695 \ , \ p^1_{t,t+2} = 0.902 \ , \ p^1_{t,t+3} = 0.8369$$

and in state 2, prices are:

$$p^2_{t,t+1} = 0.9295 \ , \ p^2_{t,t+2} = 0.902 \ , \ p^2_{t,t+3} = 0.8769$$

We will assume the same transition matrix and G_t process as above

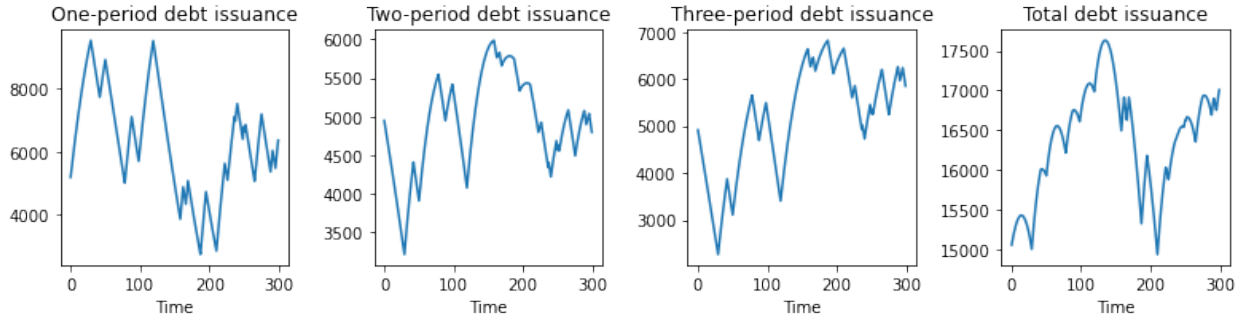
```
# New model parameters
H = 3
p1 = np.array([[0.9695], [0.902], [0.8369]])
p2 = np.array([[0.9295], [0.902], [0.8769]])
Pi = np.array([[0.9, 0.1], [0.1, 0.9]])

# 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')
ax2.plot(u[1, :])
```

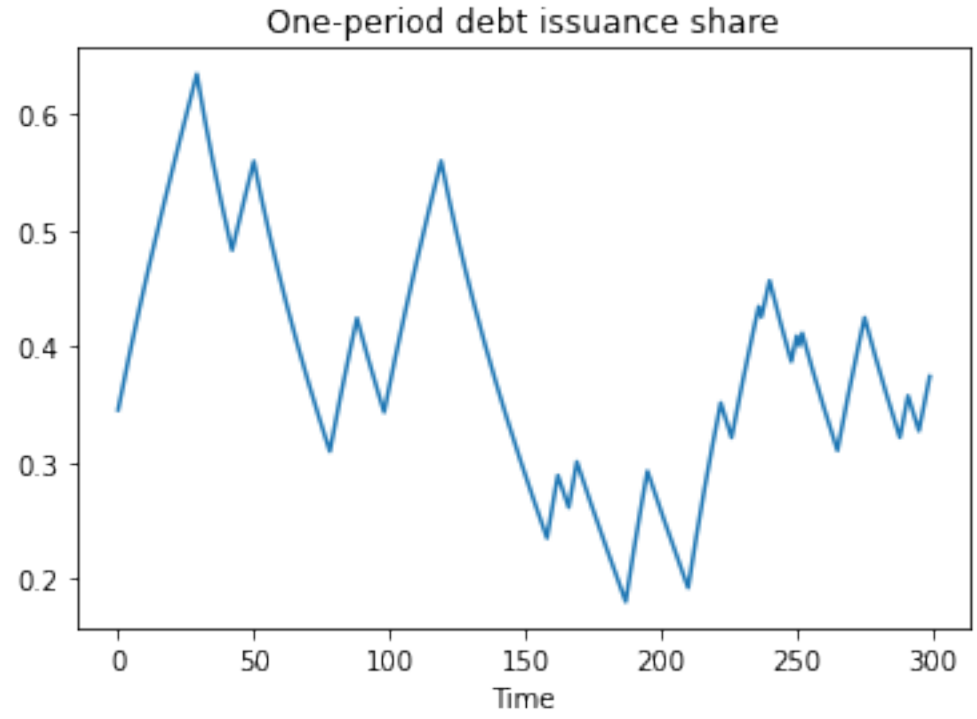
(continues on next page)

(continued from previous page)

```
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()
```



CHAPTER

TWELVE

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
```

12.1 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 [Bar99], 2003 [BM03]) that extend his classic 1979 [Bar79] model of tax smoothing.

Barro's 1979 [Bar79] 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 [Bar79] 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
%matplotlib inline
```

12.2 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.

12.3 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.

12.4 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))
S1[0, 0] = 1
```

(continues on next page)

(continued from previous page)

```
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 $p_{t+1}^t = \beta$ 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.

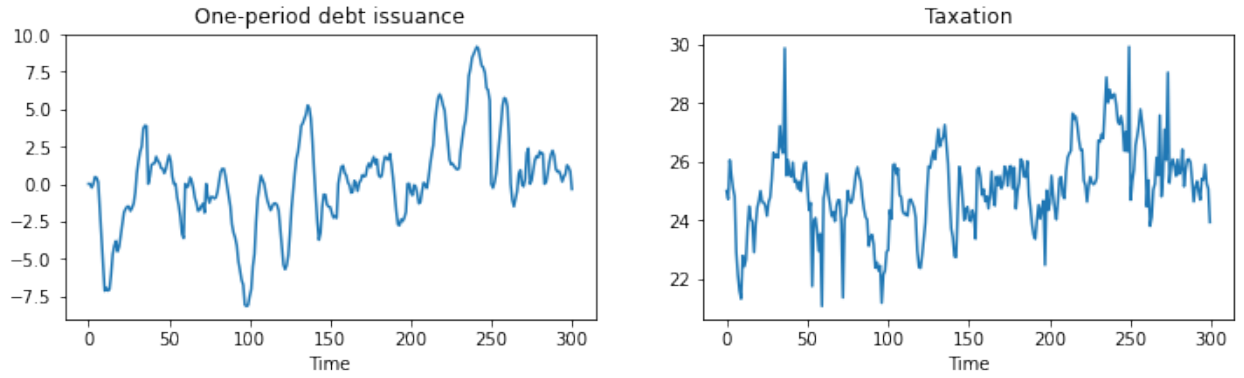
An inability to use financial markets in the next period means that the government uses those assets to lower taxation today.

```
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')
ax2.plot(tax)
ax2.set_title('Taxation')
```

(continues on next page)

(continued from previous page)

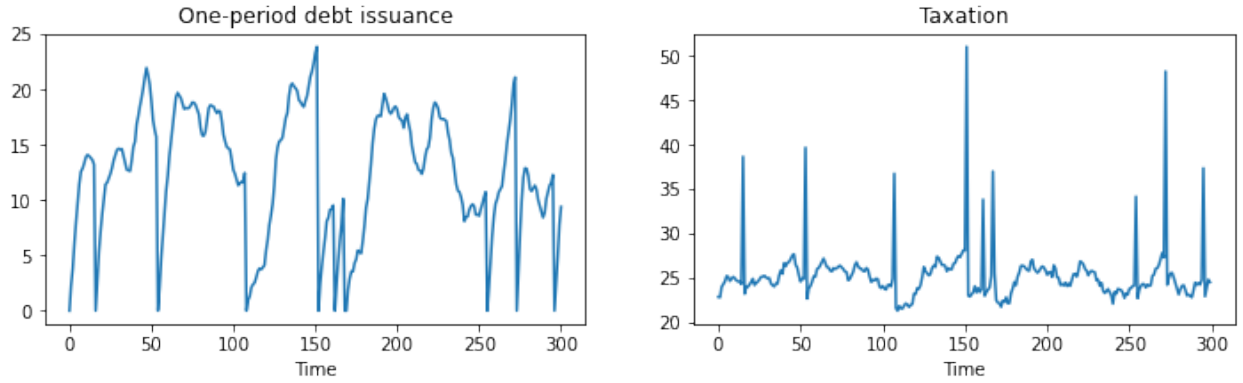
```
ax2.set_xlabel('Time')
plt.show()
```



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

THIRTEEN

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
```

13.1 Overview

In this lecture, we study optimal fiscal policy in a linear quadratic setting.

We modify a model of Robert Lucas and Nancy Stokey [LS83] 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* [Ram27] 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 [Bar79].

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
%matplotlib inline
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
```

13.1.1 Model Features

- Linear quadratic (LQ) model
- Representative household
- Stochastic dynamic programming over an infinite horizon
- Distortionary taxation

13.2 The Ramsey Problem

We begin by outlining the key assumptions regarding technology, households and the government sector.

13.2.1 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.

13.2.2 Households

Consider a representative household who chooses a path $\{\ell_t, c_t\}$ for labor and consumption to maximize

$$-\mathbb{E}\frac{1}{2}\sum_{t=0}^{\infty}\beta^t\left[(c_t-b_t)^2+\ell_t^2\right] \tag{1}$$

subject to the budget constraint

$$\mathbb{E}\sum_{t=0}^{\infty}\beta^t p_t^0\left[d_t+(1-\tau_t)\ell_t+s_t-c_t\right]=0 \tag{2}$$

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 $x^t=[x_t,x_{t-1},\dots,x_0]$ 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 (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.

13.2.3 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.

13.2.4 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.

13.2.5 Feasibility

The period-by-period feasibility restriction for this economy is

$$c_t + g_t = d_t + \ell_t \tag{3}$$

A labor-consumption process $\{\ell_t, c_t\}$ is called *feasible* if (3) holds for all t.

13.2.6 Government Budget Constraint

Where p_t^0 is again a scaled Arrow-Debreu price, the time zero government budget constraint is

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t p_t^0 (s_t + g_t - \tau_t \ell_t) = 0 \tag{4}$$

13.2.7 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 (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 (4) in terms of exogenous variables and allocations.
3. Maximize the household's objective function (1) subject to the constraint constructed in step 2 and the feasibility constraint (3).

The solution to this maximization problem pins down all quantities of interest.

13.2.8 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 (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

$$p_t^0 = \frac{b_t - c_t}{b_0 - c_0} \quad \text{and} \quad \tau_t = 1 - \frac{\ell_t}{b_t - c_t} \tag{5}$$

Substituting (5) into the government's budget constraint (4) yields

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t \left[(b_t - c_t)(s_t + g_t - \ell_t) + \ell_t^2 \right] = 0 \tag{6}$$

The Ramsey problem now amounts to maximizing (1) subject to (6) and (3).

The associated Lagrangian is

$$\mathcal{L} = \mathbb{E} \sum_{t=0}^{\infty} \beta^t \left\{ -\frac{1}{2} \left[(c_t - b_t)^2 + \ell_t^2 \right] + \lambda \left[(b_t - c_t)(\ell_t - s_t - g_t) - \ell_t^2 \right] + \mu_t [d_t + \ell_t - c_t - g_t] \right\} \tag{7}$$

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 (3) and working through the algebra, one can show that

$$\ell_t = \bar{\ell}_t - \nu m_t \quad \text{and} \quad c_t = \bar{c}_t - \nu m_t \tag{8}$$

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 (6) is $(b_t - c_t)(s_t + g_t) - (b_t - c_t)\ell_t + \ell_t^2$.

Using (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 (6), we get

$$\mathbb{E} \left\{ \sum_{t=0}^{\infty} \beta^t (b_t - \bar{c}_t)(g_t + s_t) \right\} + (\nu^2 - \nu) \mathbb{E} \left\{ \sum_{t=0}^{\infty} \beta^t 2m_t^2 \right\} = 0 \tag{9}$$

Although it might not be clear yet, we are nearly there because:

- The two expectations terms in (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 (8).
- Once we have the allocations, prices and the tax system can be derived from (5).

13.2.9 Computing the Quadratic Term

Let's consider how to obtain the term ν in (9).

If we can compute the two expected geometric sums

$$b_0 := \mathbb{E}\left\{\sum_{t=0}^{\infty}\beta^t(b_t-\bar{c}_t)(g_t+s_t)\right\} \quad \text{and} \quad a_0 := \mathbb{E}\left\{\sum_{t=0}^{\infty}\beta^t 2m_t^2\right\} \tag{10}$$

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 (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 (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

$$q(x_0) = \mathbb{E}\sum_{t=0}^{\infty}\beta^t x_t' H x_t \tag{11}$$

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 $q(x_0) = x_0'Qx_0 + v$, where

- Q is the solution to $Q = H + \beta A'QA$, and
- $v = \text{trace}\,(C'QC)\beta/(1-\beta)$

The first equation is known as a discrete Lyapunov equation and can be solved using this function.

13.2.10 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 $\{x^1, \ldots, x^N\} \subset \mathbb{R}^k$.

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, $h(x_t) = x_t'Hx_t$.

It is legitimate to pass the expectation through the sum, leading to

$$q(x_0) = \sum_{t=0}^{\infty}\beta^t(P^t h)[j] \tag{12}$$

Here

- P^t is the t-th power of the transition matrix P.
- h is, with some abuse of notation, the vector $(h(x^1), \ldots, h(x^N))$.
- $(P^t h)[j]$ indicates the j-th element of $P^t h$.

It can be shown that (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.

13.2.11 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

$$B_t := \mathbb{E}_t \sum_{j=0}^{\infty} \beta^j p_{t+j}^t (\tau_{t+j} \ell_{t+j} - g_{t+j}) \tag{13}$$

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 (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

$$B_t = (\tau_t \ell_t - g_t) + \beta E_t p_{t+1}^t B_{t+1} \tag{14}$$

Define

$$R_t^{-1} := \mathbb{E}_t \beta^j p_{t+1}^t \tag{15}$$

R_t is the gross 1-period risk-free rate for loans between t and $t + 1$.

13.2.12 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 (14) and (15) to obtain

$$\pi_{t+1} = B_{t+1} - \frac{1}{\beta E_t p_{t+1}^t} \left[\beta E_t p_{t+1}^t B_{t+1}\right]$$

or

$$\pi_{t+1} = B_{t+1} - \tilde{E}_t B_{t+1} \tag{16}$$

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

$$m_{t+1}^t = \frac{p_{t+1}^t}{E_t p_{t+1}^t}$$

It follows from equation (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 [Bar79], 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.

13.3 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

```
# 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
                  '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
```

(continues on next page)

(continued from previous page)

```
        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

    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)
```

(continues on next page)

(continued from previous page)

```
        H = (Sb - Sd + Sg).T @ (Sg + Ss)
        b0 = 0.5 * var_quadratic_sum(A, C, H, β, x0)

    # Test that ν 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:
        ν = 0.5 * (a0 - sqrt(disc)) / a0
    else:
        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
```

(continues on next page)

(continued from previous page)

```
    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.
    """

    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):
    """
```

(continues on next page)

(continued from previous page)

```
    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)
    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()
```

13.3.1 Comments on the Code

The function `var_quadratic_sum` imported from `quadsums` is for computing the value of (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.

13.4 Examples

Let's look at two examples of usage.

13.4.1 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

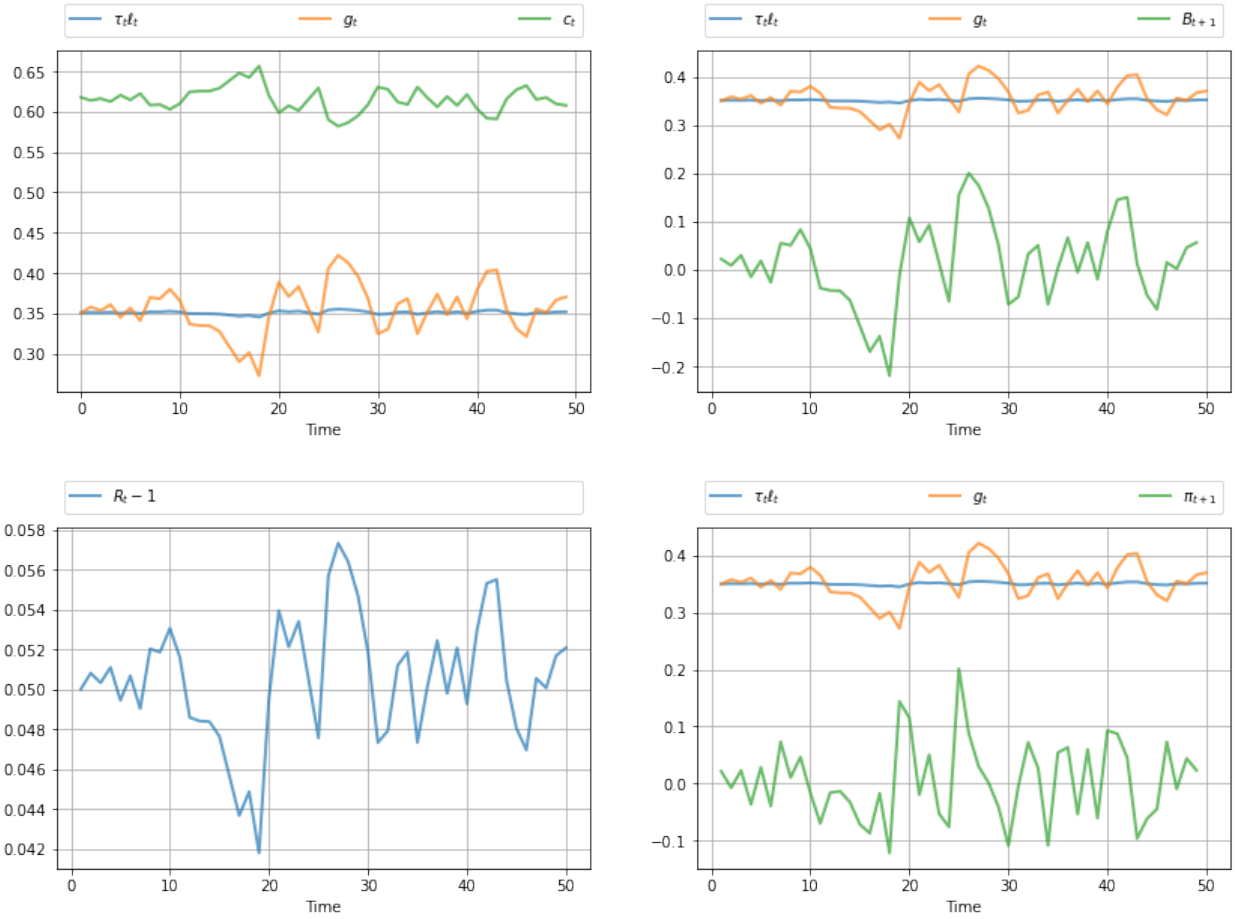
Government spending evolves according to

$$g_{t+1} - \mu_g = \rho(g_t - \mu_g) + C_g w_{g,t+1}$$

with $\rho = 0.7$, $\mu_g = 0.35$ and $C_g = \mu_g\sqrt{1-\rho^2}/10$.

Here's the code

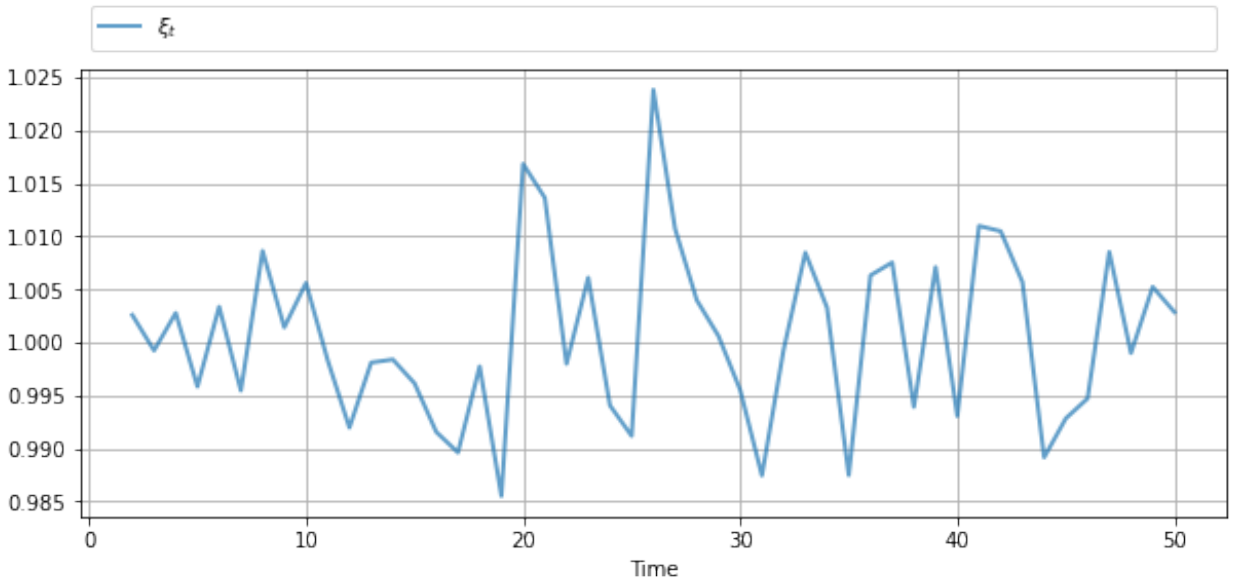
```
# == Parameters == #
β = 1 / 1.05
ρ, mg = .7, .35
A = eye(2)
A[0, :] = ρ, mg * (1-ρ)
C = np.zeros((2, 1))
C[0, 0] = np.sqrt(1 - ρ**2) * mg / 10
Sg = np.array((1, 0)).reshape(1, 2)
Sd = np.array((0, 0)).reshape(1, 2)
Sb = np.array((0, 2.135)).reshape(1, 2)
Ss = np.array((0, 0)).reshape(1, 2)

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)
```

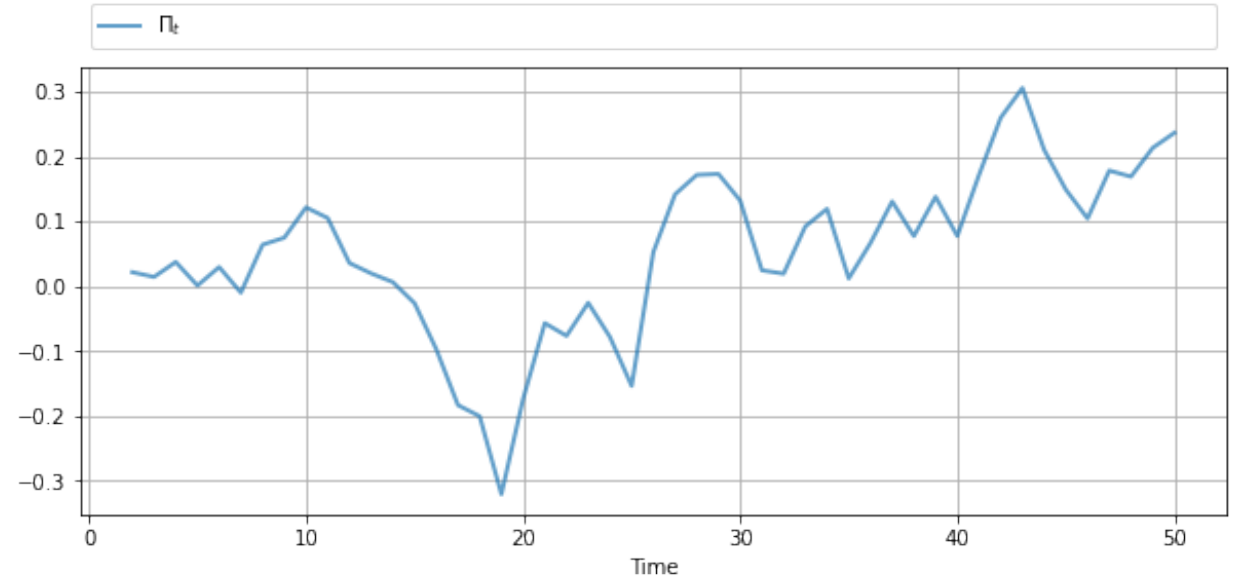


The legends on the figures indicate the variables being tracked.

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.

13.4.2 The Discrete Case

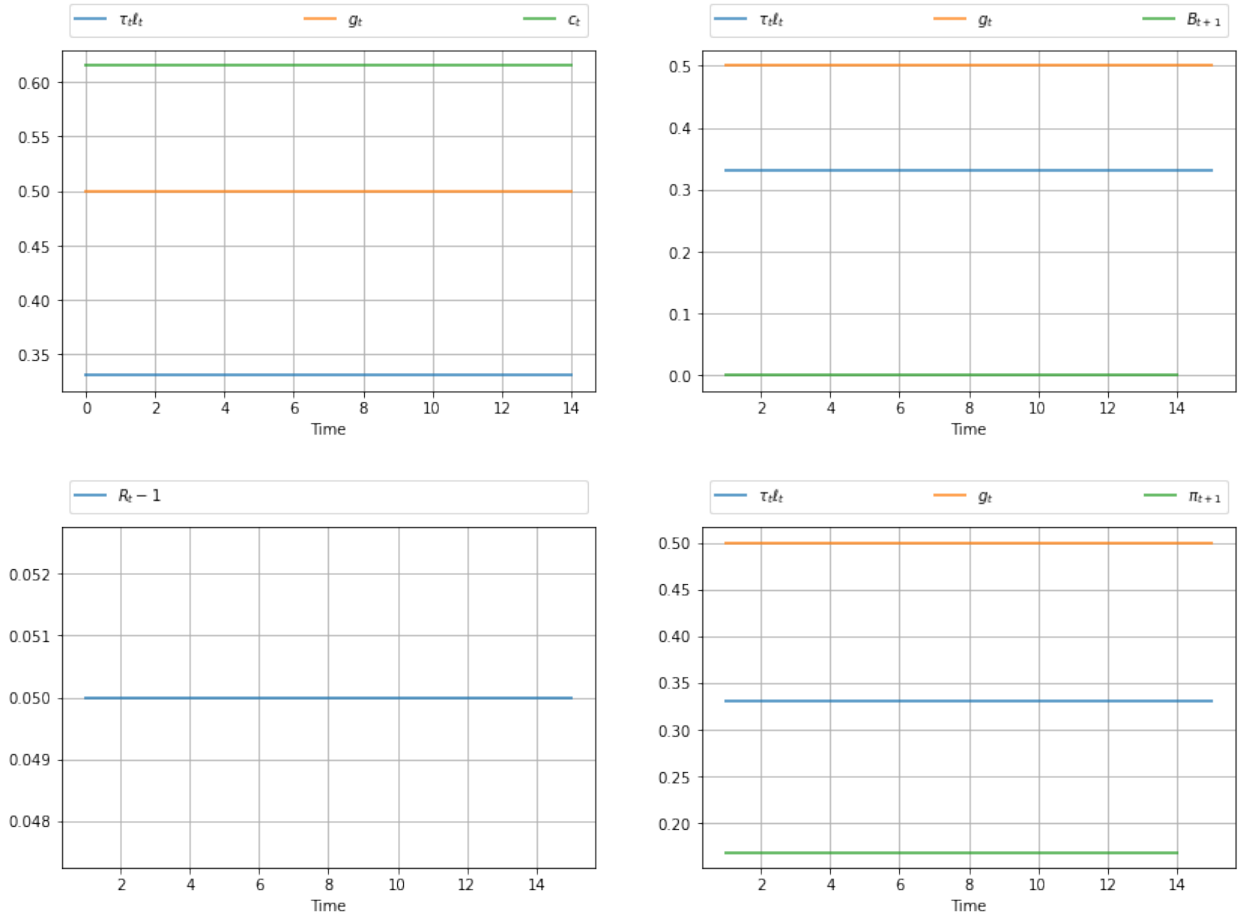
Our second example adopts a discrete Markov specification for the exogenous process

```
# == 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]])
```

(continues on next page)

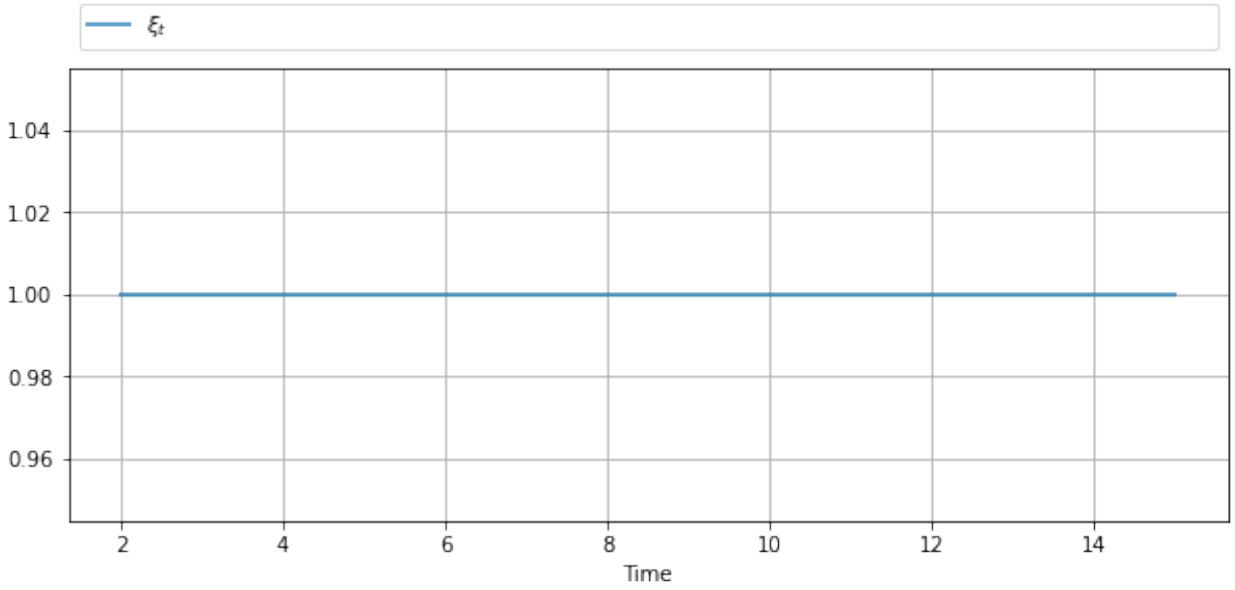
(continued from previous page)

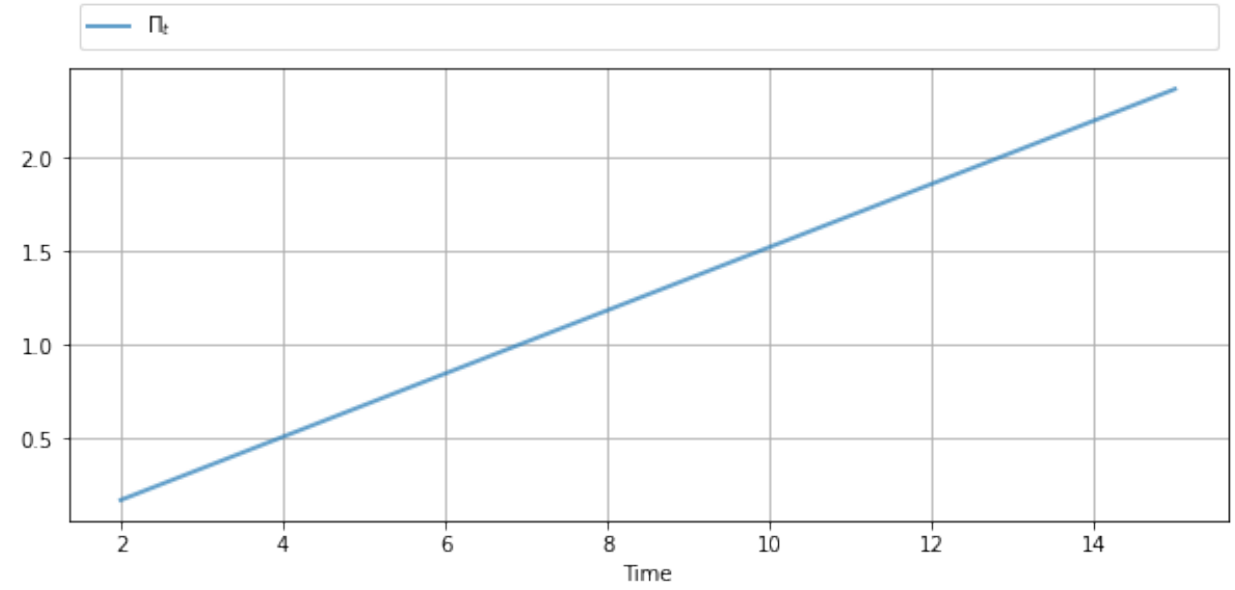
```
# Possible states of the world
# Each column is a state of the world. The rows are [g d b s 1]
x_vals = np.array([[0.5, 0.5, 0.25],
                   [0.0, 0.0,  0.0],
                   [2.2, 2.2,  2.2],
                   [0.0, 0.0,  0.0],
                   [1.0, 1.0,  1.0]])

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)
```



The call `gen_fig_2(path)` generates

```
gen_fig_2(path)
```





See the original manuscript for comments and interpretation.

13.5 Exercises

13.5.1 Exercise 1

Modify the VAR example *given above*, setting

$$g_{t+1} - \mu_g = \rho(g_{t-3} - \mu_g) + C_g w_{g,t+1}$$

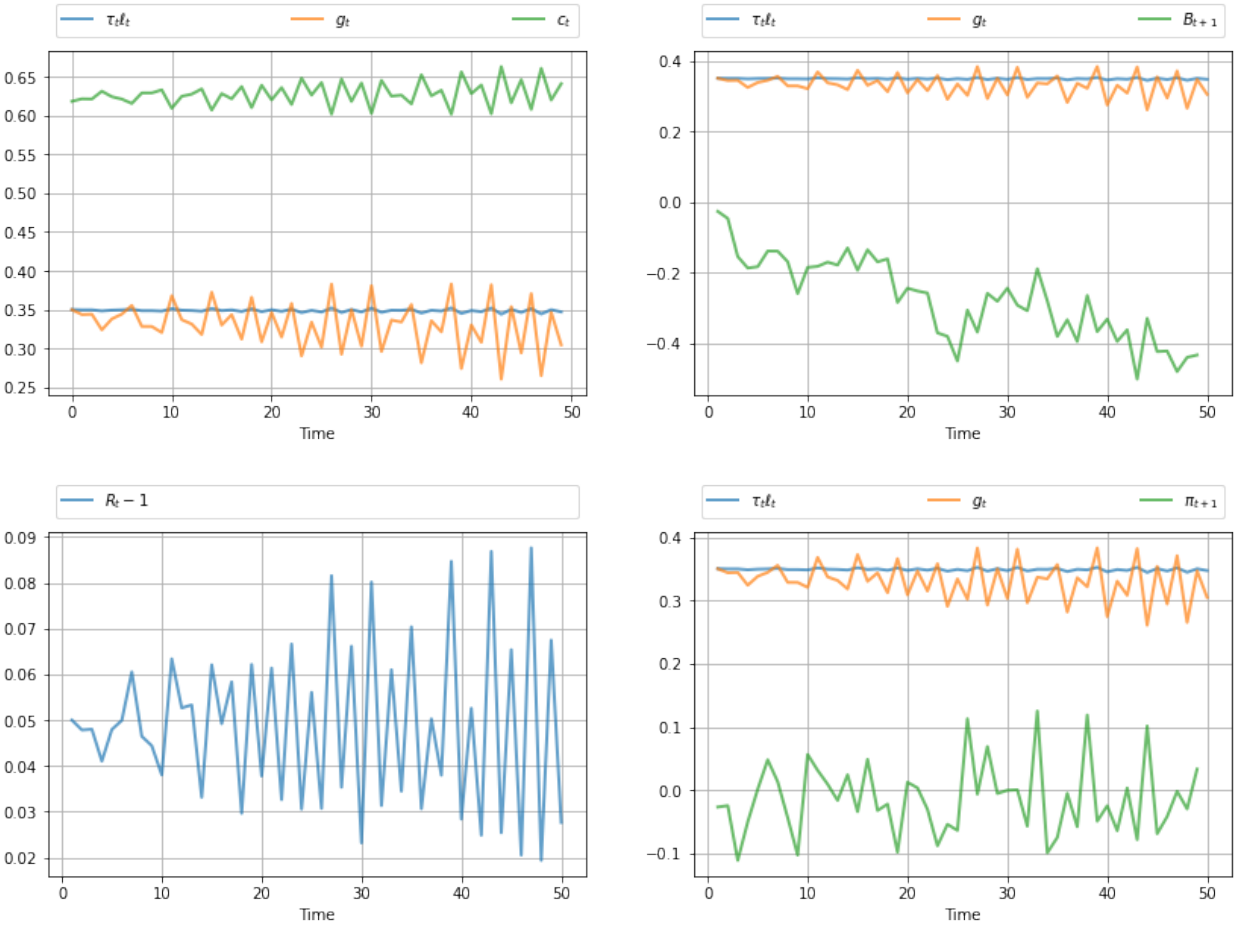
with $\rho = 0.95$ and $C_g = 0.7\sqrt{1-\rho^2}$.

Produce the corresponding figures.

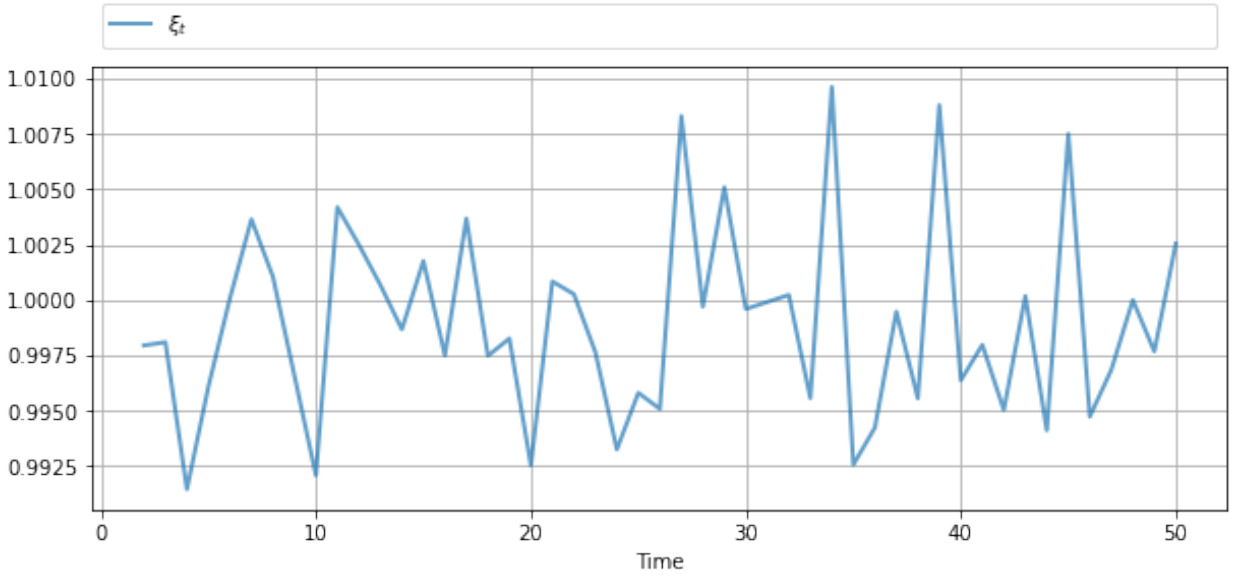
13.6 Solutions

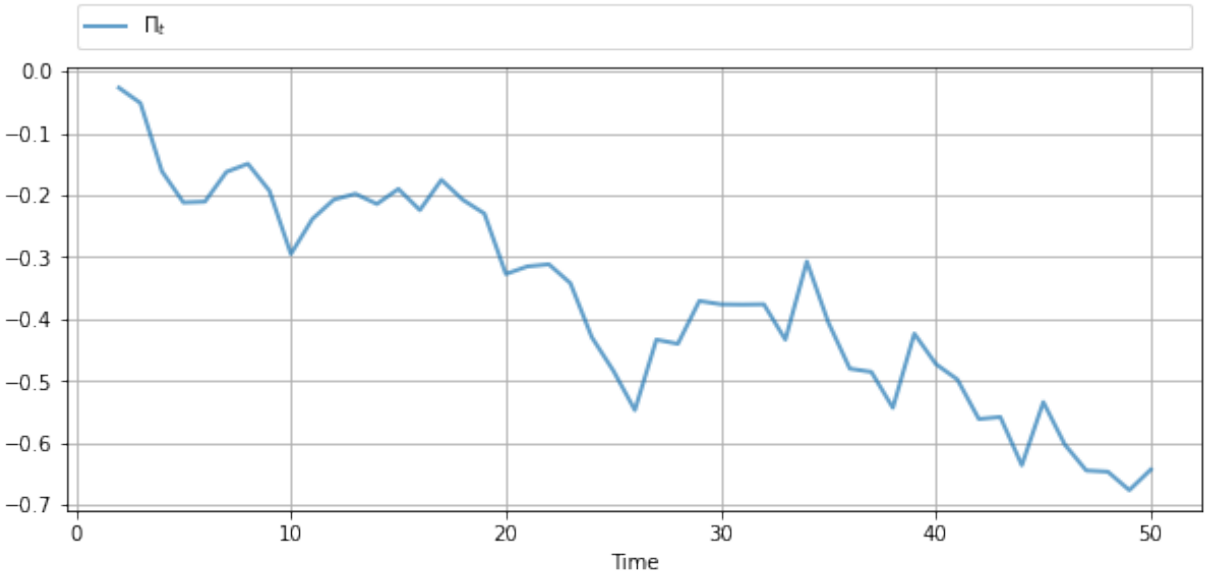
13.6.1 Exercise 1

```
# == 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
1.0100
1.0075
1.0050
1.0025
1.0000
0.9975
0.9950
0.9925
0
10
20
30
40
50
Time


Π_t
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
0
10
20
30
40
50
Time