Part IX

Asset Pricing and Finance

CHAPTER

FIFTYNINE

ASSET PRICING: FINITE STATE MODELS

Contents

> "A little knowledge of geometric series goes a long way" – Robert E. Lucas, Jr.
>
> "Asset pricing is all about covariances" – Lars Peter Hansen

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

```
!conda install -y quantecon
```

59.1 Overview

An asset is a claim on one or more future payoffs.

The spot price of an asset depends primarily on

- the anticipated dynamics for the stream of income accruing to the owners
- attitudes to risk
- rates of time preference

In this lecture, we consider some standard pricing models and dividend stream specifications.

We study how prices and dividend-price ratios respond in these different scenarios.

We also look at creating and pricing *derivative* assets by repackaging income streams.

Key tools for the lecture are

- formulas for predicting future values of functions of a Markov state
- a formula for predicting the discounted sum of future values of a Markov state

Let's start with some imports:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
import quantecon as qe
from numpy.linalg import eigvals, solve
```

59.2 Pricing Models

In what follows let $\{d_t\}_{t \geq 0}$ be a stream of dividends

- A time-t **cum-dividend** asset is a claim to the stream $d_t, d_{t+1}, ...$
- A time-t **ex-dividend** asset is a claim to the stream $d_{t+1}, d_{t+2}, ...$

Let's look at some equations that we expect to hold for prices of assets under ex-dividend contracts (we will consider cum-dividend pricing in the exercises).

59.2.1 Risk-Neutral Pricing

Our first scenario is risk-neutral pricing.

Let $\beta = 1/(1 + \rho)$ be an intertemporal discount factor, where ρ is the rate at which agents discount the future.

The basic risk-neutral asset pricing equation for pricing one unit of an ex-dividend asset is

$$p_t = \beta \mathbb{E}_t [d_{t+1} + p_{t+1}] \tag{1}$$

This is a simple "cost equals expected benefit" relationship.

Here $\mathbb{E}_t[y]$ denotes the best forecast of y, conditioned on information available at time t.

59.2.2 Pricing with Random Discount Factor

What happens if for some reason traders discount payouts differently depending on the state of the world?

Michael Harrison and David Kreps [HK79] and Lars Peter Hansen and Scott Richard [HR87] showed that in quite general settings the price of an ex-dividend asset obeys

$$p_t = \mathbb{E}_t \left[m_{t+1} (d_{t+1} + p_{t+1}) \right] \tag{2}$$

for some **stochastic discount factor** m_{t+1}.

The fixed discount factor β in (1) has been replaced by the random variable m_{t+1}.

The way anticipated future payoffs are evaluated can now depend on various random outcomes.

One example of this idea is that assets that tend to have good payoffs in bad states of the world might be regarded as more valuable.

This is because they pay well when funds are more urgently wanted.

We give examples of how the stochastic discount factor has been modeled below.

59.2.3 Asset Pricing and Covariances

Recall that, from the definition of a conditional covariance $\text{cov}_t(x_{t+1}, y_{t+1})$, we have

$$\mathbb{E}_t(x_{t+1}y_{t+1}) = \text{cov}_t(x_{t+1}, y_{t+1}) + \mathbb{E}_t x_{t+1}\mathbb{E}_t y_{t+1} \tag{3}$$

If we apply this definition to the asset pricing equation (2) we obtain

$$p_t = \mathbb{E}_t m_{t+1}\mathbb{E}_t(d_{t+1} + p_{t+1}) + \text{cov}_t(m_{t+1}, d_{t+1} + p_{t+1}) \tag{4}$$

It is useful to regard equation (4) as a generalization of equation (1)

- In equation (1), the stochastic discount factor $m_{t+1} = \beta$, a constant.
- In equation (1), the covariance term $\text{cov}_t(m_{t+1}, d_{t+1} + p_{t+1})$ is zero because $m_{t+1} = \beta$.
- In equation (1), $\mathbb{E}_t m_{t+1}$ can be interpreted as the reciprocal of the one-period risk-free gross interest rate.
- When m_{t+1} is covaries more negatively with the payout $p_{t+1} + d_{t+1}$, the price of the asset is lower.

Equation (4) asserts that the covariance of the stochastic discount factor with the one period payout $d_{t+1} + p_{t+1}$ is an important determinant of the price p_t.

We give examples of some models of stochastic discount factors that have been proposed later in this lecture and also in a later lecture.

59.2.4 The Price-Dividend Ratio

Aside from prices, another quantity of interest is the **price-dividend ratio** $v_t := p_t/d_t$.

Let's write down an expression that this ratio should satisfy.

We can divide both sides of (2) by d_t to get

$$v_t = \mathbb{E}_t \left[m_{t+1} \frac{d_{t+1}}{d_t}(1 + v_{t+1}) \right] \tag{5}$$

Below we'll discuss the implication of this equation.

59.3 Prices in the Risk-Neutral Case

What can we say about price dynamics on the basis of the models described above?

The answer to this question depends on

1. the process we specify for dividends
2. the stochastic discount factor and how it correlates with dividends

For now we'll study the risk-neutral case in which the stochastic discount factor is constant.

We'll focus on how the asset prices depends on the dividend process.

59.3.1 Example 1: Constant Dividends

The simplest case is risk-neutral pricing in the face of a constant, non-random dividend stream $d_t = d > 0$.

Removing the expectation from (1) and iterating forward gives

$$\begin{aligned} p_t &= \beta(d + p_{t+1}) \\ &= \beta(d + \beta(d + p_{t+2})) \\ &\quad \vdots \\ &= \beta(d + \beta d + \beta^2 d + \cdots + \beta^{k-2} d + \beta^{k-1} p_{t+k}) \end{aligned}$$

Unless prices explode in the future, this sequence converges to

$$\bar{p} := \frac{\beta d}{1 - \beta} \tag{6}$$

This price is the equilibrium price in the constant dividend case.

Indeed, simple algebra shows that setting $p_t = \bar{p}$ for all t satisfies the equilibrium condition $p_t = \beta(d + p_{t+1})$.

59.3.2 Example 2: Dividends with Deterministic Growth Paths

Consider a growing, non-random dividend process $d_{t+1} = g d_t$ where $0 < g\beta < 1$.

While prices are not usually constant when dividends grow over time, the price dividend-ratio might be.

If we guess this, substituting $v_t = v$ into (5) as well as our other assumptions, we get $v = \beta g(1 + v)$.

Since $\beta g < 1$, we have a unique positive solution:

$$v = \frac{\beta g}{1 - \beta g}$$

The price is then

$$p_t = \frac{\beta g}{1 - \beta g} d_t$$

If, in this example, we take $g = 1 + \kappa$ and let $\rho := 1/\beta - 1$, then the price becomes

$$p_t = \frac{1 + \kappa}{\rho - \kappa} d_t$$

This is called the *Gordon formula*.

59.3.3 Example 3: Markov Growth, Risk-Neutral Pricing

Next, we consider a dividend process

$$d_{t+1} = g_{t+1} d_t \tag{7}$$

The stochastic growth factor $\{g_t\}$ is given by

$$g_t = g(X_t), \quad t = 1, 2, ...$$

where

1. $\{X_t\}$ is a finite Markov chain with state space S and transition probabilities

$$P(x, y) := \mathbb{P}\{X_{t+1} = y \,|\, X_t = x\} \qquad (x, y \in S)$$

2. g is a given function on S taking positive values

You can think of

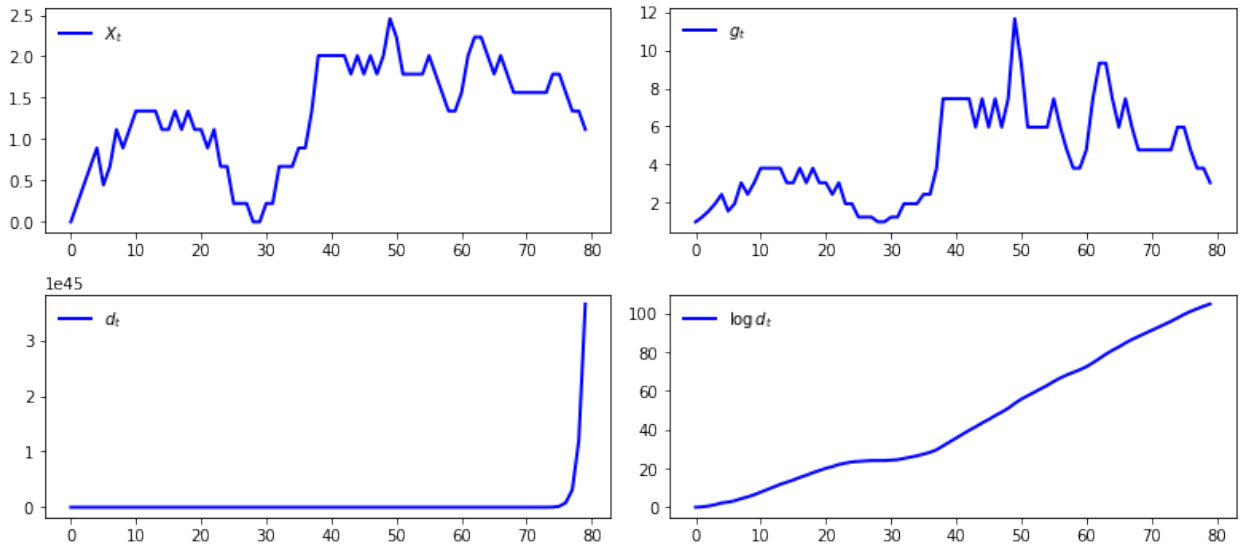
- S as n possible "states of the world" and X_t as the current state.
- g as a function that maps a given state X_t into a growth of dividends factor $g_t = g(X_t)$.
- $\ln g_t = \ln(d_{t+1}/d_t)$ is the growth rate of dividends.

(For a refresher on notation and theory for finite Markov chains see *this lecture*)

The next figure shows a simulation, where

- $\{X_t\}$ evolves as a discretized AR1 process produced using *Tauchen's method*.
- $g_t = \exp(X_t)$, so that $\ln g_t = X_t$ is the growth rate.

```
mc = qe.tauchen(0.96, 0.25, n=25)
sim_length = 80

x_series = mc.simulate(sim_length, init=np.median(mc.state_values))
g_series = np.exp(x_series)
d_series = np.cumprod(g_series) # Assumes d_0 = 1

series = [x_series, g_series, d_series, np.log(d_series)]
labels = ['$X_t$', '$g_t$', '$d_t$', r'$\log \, d_t$']

fig, axes = plt.subplots(2, 2)
for ax, s, label in zip(axes.flatten(), series, labels):
    ax.plot(s, 'b-', lw=2, label=label)
    ax.legend(loc='upper left', frameon=False)
plt.tight_layout()
plt.show()
```



Pricing

To obtain asset prices in this setting, let's adapt our analysis from the case of deterministic growth.

In that case, we found that v is constant.

This encourages us to guess that, in the current case, v_t is constant given the state X_t.

In other words, we are looking for a fixed function v such that the price-dividend ratio satisfies $v_t = v(X_t)$.

We can substitute this guess into (5) to get

$$v(X_t) = \beta \mathbb{E}_t[g(X_{t+1})(1 + v(X_{t+1}))]$$

If we condition on $X_t = x$, this becomes

$$v(x) = \beta \sum_{y \in S} g(y)(1 + v(y))P(x, y)$$

or

$$v(x) = \beta \sum_{y \in S} K(x, y)(1 + v(y)) \quad \text{where} \quad K(x, y) := g(y)P(x, y) \tag{8}$$

Suppose that there are n possible states $x_1, \ldots, x_n$.

We can then think of (8) as n stacked equations, one for each state, and write it in matrix form as

$$v = \beta K(\mathbb{1} + v) \tag{9}$$

Here

- v is understood to be the column vector $(v(x_1), \ldots, v(x_n))'$.
- K is the matrix $(K(x_i, x_j))_{1 \leq i,j \leq n}$.
- $\mathbb{1}$ is a column vector of ones.

When does (9) have a unique solution?

From the *Neumann series lemma* and Gelfand's formula, this will be the case if βK has spectral radius strictly less than one.

In other words, we require that the eigenvalues of K be strictly less than β^{-1} in modulus.

The solution is then

$$v = (I - \beta K)^{-1} \beta K \mathbb{1} \tag{10}$$

59.3.4 Code

Let's calculate and plot the price-dividend ratio at a set of parameters.

As before, we'll generate $\{X_t\}$ as a *discretized AR1 process* and set $g_t = \exp(X_t)$.

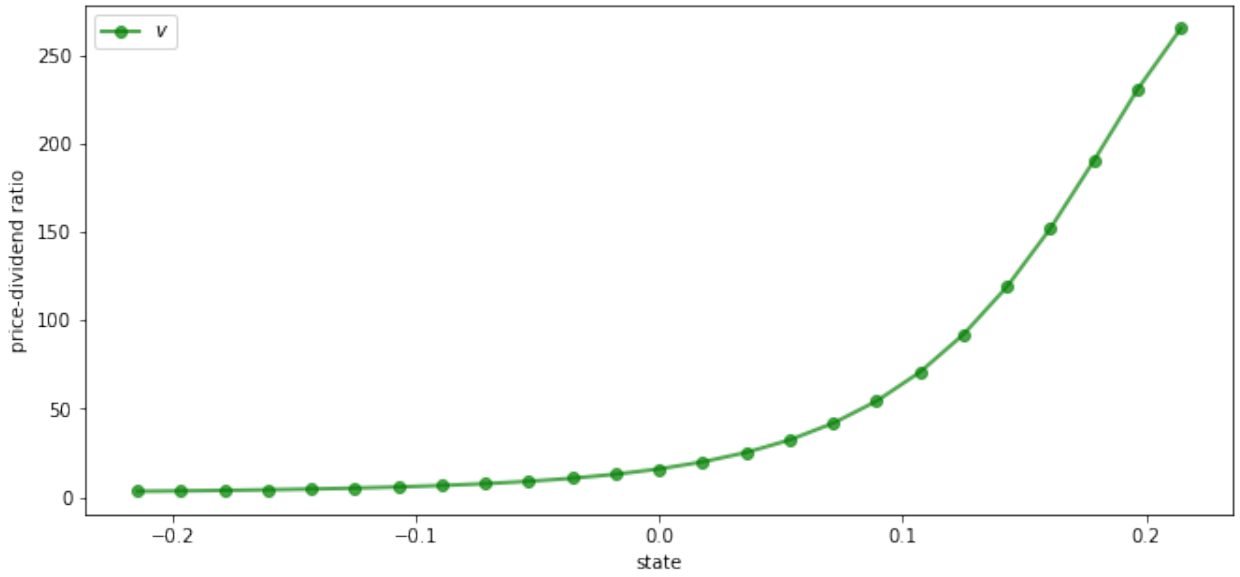
Here's the code, including a test of the spectral radius condition

```
n = 25  # Size of state space
β = 0.9
mc = qe.tauchen(0.96, 0.02, n=n)

K = mc.P * np.exp(mc.state_values)
```

(continues on next page)

(continued from previous page)

```
warning_message = "Spectral radius condition fails"
assert np.max(np.abs(eigvals(K))) < 1 / β,  warning_message

I = np.identity(n)
v = solve(I - β * K, β * K @ np.ones(n))

fig, ax = plt.subplots()
ax.plot(mc.state_values, v, 'g-o', lw=2, alpha=0.7, label='$v$')
ax.set_ylabel("price-dividend ratio")
ax.set_xlabel("state")
ax.legend(loc='upper left')
plt.show()
```



Why does the price-dividend ratio increase with the state?

The reason is that this Markov process is positively correlated, so high current states suggest high future states.

Moreover, dividend growth is increasing in the state.

The anticipation of high future dividend growth leads to a high price-dividend ratio.

59.4 Risk Aversion and Asset Prices

Now let's turn to the case where agents are risk averse.

We'll price several distinct assets, including

- An endowment stream
- A consol (a type of bond issued by the UK government in the 19th century)
- Call options on a consol

59.4.1 Pricing a Lucas Tree

Let's start with a version of the celebrated asset pricing model of Robert E. Lucas, Jr. [Luc78].

As in [Luc78], suppose that the stochastic discount factor takes the form

$$m_{t+1} = \beta \frac{u'(c_{t+1})}{u'(c_t)} \tag{11}$$

where u is a concave utility function and c_t is time t consumption of a representative consumer.

(A derivation of this expression is given in a later lecture)

Assume the existence of an endowment that follows growth process (7).

The asset being priced is a claim on the endowment process.

Following [Luc78], suppose further that in equilibrium, consumption is equal to the endowment, so that $d_t = c_t$ for all t.

For utility, we'll assume the **constant relative risk aversion** (CRRA) specification

$$u(c) = \frac{c^{1-\gamma}}{1-\gamma} \text{ with } \gamma > 0 \tag{12}$$

When $\gamma = 1$ we let $u(c) = \ln c$.

Inserting the CRRA specification into (11) and using $c_t = d_t$ gives

$$m_{t+1} = \beta \left(\frac{c_{t+1}}{c_t} \right)^{-\gamma} = \beta g_{t+1}^{-\gamma} \tag{13}$$

Substituting this into (5) gives the price-dividend ratio formula

$$v(X_t) = \beta \mathbb{E}_t \left[g(X_{t+1})^{1-\gamma} (1 + v(X_{t+1})) \right]$$

Conditioning on $X_t = x$, we can write this as

$$v(x) = \beta \sum_{y \in S} g(y)^{1-\gamma} (1 + v(y)) P(x, y)$$

If we let

$$J(x, y) := g(y)^{1-\gamma} P(x, y)$$

then we can rewrite in vector form as

$$v = \beta J (\mathbb{1} + v)$$

Assuming that the spectral radius of J is strictly less than β^{-1}, this equation has the unique solution

$$v = (I - \beta J)^{-1} \beta J \mathbb{1} \tag{14}$$

We will define a function tree_price to solve for v given parameters stored in the class AssetPriceModel

```
class AssetPriceModel:
    """
    A class that stores the primitives of the asset pricing model.

    Parameters
    ----------
```

(continues on next page)

(continued from previous page)

```
    β : scalar, float
        Discount factor
    mc : MarkovChain
        Contains the transition matrix and set of state values for the state
        process
    γ : scalar(float)
        Coefficient of risk aversion
    g : callable
        The function mapping states to growth rates

    """
    def __init__(self, β=0.96, mc=None, γ=2.0, g=np.exp):
        self.β, self.γ = β, γ
        self.g = g

        # A default process for the Markov chain
        if mc is None:
            self.ρ = 0.9
            self.σ = 0.02
            self.mc = qe.tauchen(self.ρ, self.σ, n=25)
        else:
            self.mc = mc

        self.n = self.mc.P.shape[0]

    def test_stability(self, Q):
        """
        Stability test for a given matrix Q.
        """
        sr = np.max(np.abs(eigvals(Q)))
        if not sr < 1 / self.β:
            msg = f"Spectral radius condition failed with radius = {sr}"
            raise ValueError(msg)


def tree_price(ap):
    """
    Computes the price-dividend ratio of the Lucas tree.

    Parameters
    ----------
    ap: AssetPriceModel
        An instance of AssetPriceModel containing primitives

    Returns
    -------
    v : array_like(float)
        Lucas tree price-dividend ratio

    """
    # Simplify names, set up matrices
    β, γ, P, y = ap.β, ap.γ, ap.mc.P, ap.mc.state_values
    J = P * ap.g(y)**(1 - γ)

    # Make sure that a unique solution exists
    ap.test_stability(J)
```

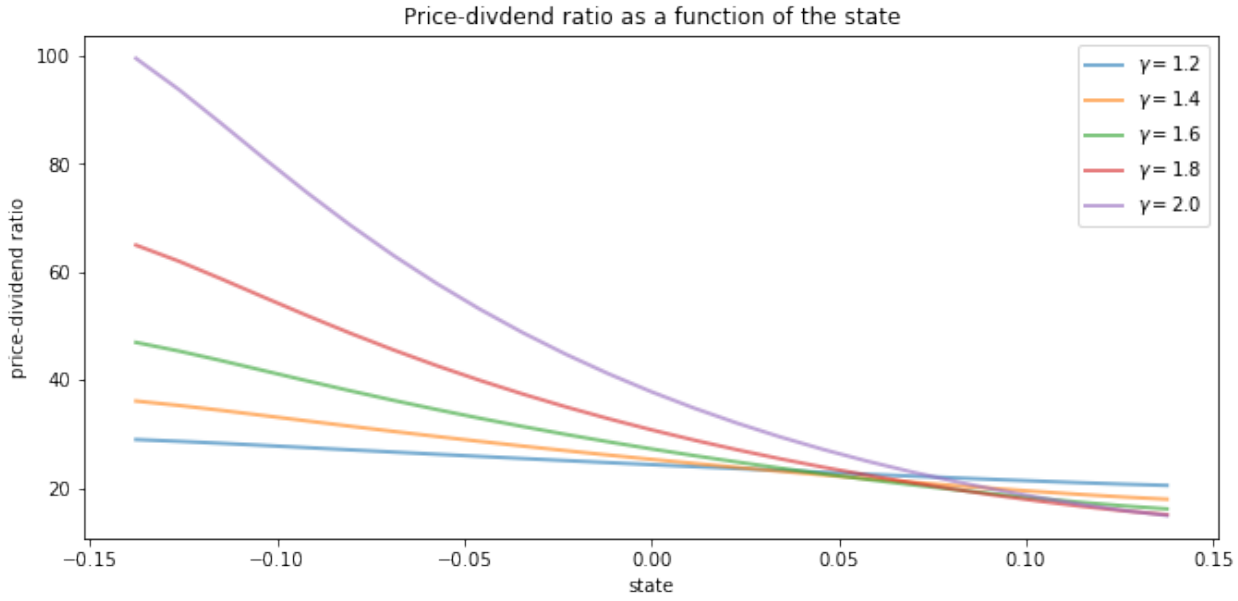
(continues on next page)

(continued from previous page)

```
    # Compute v
    I = np.identity(ap.n)
    Ones = np.ones(ap.n)
    v = solve(I - β * J, β * J @ Ones)

    return v
```

Here's a plot of v as a function of the state for several values of γ, with a positively correlated Markov process and $g(x) = \exp(x)$

```
ys = [1.2, 1.4, 1.6, 1.8, 2.0]
ap = AssetPriceModel()
states = ap.mc.state_values

fig, ax = plt.subplots()

for γ in ys:
    ap.γ = γ
    v = tree_price(ap)
    ax.plot(states, v, lw=2, alpha=0.6, label=rf"$\gamma = {γ}$")

ax.set_title('Price-divdend ratio as a function of the state')
ax.set_ylabel("price-dividend ratio")
ax.set_xlabel("state")
ax.legend(loc='upper right')
plt.show()
```



Notice that v is decreasing in each case.

This is because, with a positively correlated state process, higher states suggest higher future consumption growth.

In the stochastic discount factor (13), higher growth decreases the discount factor, lowering the weight placed on future returns.

Special Cases

In the special case $\gamma = 1$, we have $J = P$.

Recalling that $P^i \mathbb{1} = \mathbb{1}$ for all i and applying *Neumann's geometric series lemma*, we are led to

$$v = \beta(I - \beta P)^{-1}\mathbb{1} = \beta \sum_{i=0}^{\infty} \beta^i P^i \mathbb{1} = \beta \frac{1}{1-\beta}\mathbb{1}$$

Thus, with log preferences, the price-dividend ratio for a Lucas tree is constant.

Alternatively, if $\gamma = 0$, then $J = K$ and we recover the risk-neutral solution (10).

This is as expected, since $\gamma = 0$ implies $u(c) = c$ (and hence agents are risk-neutral).

59.4.2 A Risk-Free Consol

Consider the same pure exchange representative agent economy.

A risk-free consol promises to pay a constant amount $\zeta > 0$ each period.

Recycling notation, let p_t now be the price of an ex-coupon claim to the consol.

An ex-coupon claim to the consol entitles the owner at the end of period t to

- ζ in period $t + 1$, plus
- the right to sell the claim for p_{t+1} next period

The price satisfies (2) with $d_t = \zeta$, or

$$p_t = \mathbb{E}_t \left[m_{t+1}(\zeta + p_{t+1}) \right]$$

We maintain the stochastic discount factor (13), so this becomes

$$p_t = \mathbb{E}_t \left[\beta g_{t+1}^{-\gamma}(\zeta + p_{t+1}) \right] \tag{15}$$

Guessing a solution of the form $p_t = p(X_t)$ and conditioning on $X_t = x$, we get

$$p(x) = \beta \sum_{y \in S} g(y)^{-\gamma}(\zeta + p(y))P(x, y)$$

Letting $M(x, y) = P(x, y)g(y)^{-\gamma}$ and rewriting in vector notation yields the solution

$$p = (I - \beta M)^{-1}\beta M \zeta \mathbb{1} \tag{16}$$

The above is implemented in the function consol_price.

```
def consol_price(ap, ζ):
    """
    Computes price of a consol bond with payoff ζ

    Parameters
    ----------
    ap: AssetPriceModel
        An instance of AssetPriceModel containing primitives

    ζ : scalar(float)
        Coupon of the console
```

(continues on next page)

(continued from previous page)

```
    Returns
    -------
    p : array_like(float)
        Console bond prices

    """
    # Simplify names, set up matrices
    β, γ, P, y = ap.β, ap.γ, ap.mc.P, ap.mc.state_values
    M = P * ap.g(y)**(- γ)

    # Make sure that a unique solution exists
    ap.test_stability(M)

    # Compute price
    I = np.identity(ap.n)
    Ones = np.ones(ap.n)
    p = solve(I - β * M, β * ζ * M @ Ones)

    return p
```

59.4.3 Pricing an Option to Purchase the Consol

Let's now price options of varying maturities.

We'll study an option that gives the owner the right to purchase a consol at a price p_S.

An Infinite Horizon Call Option

We want to price an infinite horizon option to purchase a consol at a price p_S.

The option entitles the owner at the beginning of a period either

1. to purchase the bond at price p_S now, or
2. not to exercise the option to purchase the asset now but to retain the right to exercise it later

Thus, the owner either *exercises* the option now or chooses *not to exercise* and wait until next period.

This is termed an infinite-horizon *call option* with *strike price* p_S.

The owner of the option is entitled to purchase the consol at price p_S at the beginning of any period, after the coupon has been paid to the previous owner of the bond.

The fundamentals of the economy are identical with the one above, including the stochastic discount factor and the process for consumption.

Let $w(X_t, p_S)$ be the value of the option when the time t growth state is known to be X_t but *before* the owner has decided whether to exercise the option at time t (i.e., today).

Recalling that $p(X_t)$ is the value of the consol when the initial growth state is X_t, the value of the option satisfies

$$w(X_t, p_S) = \max \left\{ \beta \, \mathbb{E}_t \frac{u'(c_{t+1})}{u'(c_t)} w(X_{t+1}, p_S), \; p(X_t) - p_S \right\}$$

The first term on the right is the value of waiting, while the second is the value of exercising now.

We can also write this as

$$w(x, p_S) = \max \left\{ \beta \sum_{y \in S} P(x, y) g(y)^{-\gamma} w(y, p_S),\ p(x) - p_S \right\} \tag{17}$$

With $M(x, y) = P(x, y)g(y)^{-\gamma}$ and w as the vector of values $(w(x_i), p_S)_{i=1}^n$, we can express (17) as the nonlinear vector equation

$$w = \max\{\beta M w,\ p - p_S \mathbb{1}\} \tag{18}$$

To solve (18), form the operator T mapping vector w into vector Tw via

$$Tw = \max\{\beta M w,\ p - p_S \mathbb{1}\}$$

Start at some initial w and iterate with T to convergence .

We can find the solution with the following function call_option

```
def call_option(ap, ζ, p_s, ϵ=1e-7):
    """
    Computes price of a call option on a consol bond.

    Parameters
    ----------
    ap: AssetPriceModel
        An instance of AssetPriceModel containing primitives

    ζ : scalar(float)
        Coupon of the console

    p_s : scalar(float)
        Strike price

    ϵ : scalar(float), optional(default=1e-8)
        Tolerance for infinite horizon problem

    Returns
    -------
    w : array_like(float)
        Infinite horizon call option prices

    """
    # Simplify names, set up matrices
    β, γ, P, y = ap.β, ap.γ, ap.mc.P, ap.mc.state_values
    M = P * ap.g(y)**(- γ)

    # Make sure that a unique consol price exists
    ap.test_stability(M)

    # Compute option price
    p = consol_price(ap, ζ)
    w = np.zeros(ap.n)
    error = ϵ + 1
    while error > ϵ:
        # Maximize across columns
        w_new = np.maximum(β * M @ w, p - p_s)
        # Find maximal difference of each component and update
        error = np.amax(np.abs(w - w_new))
```

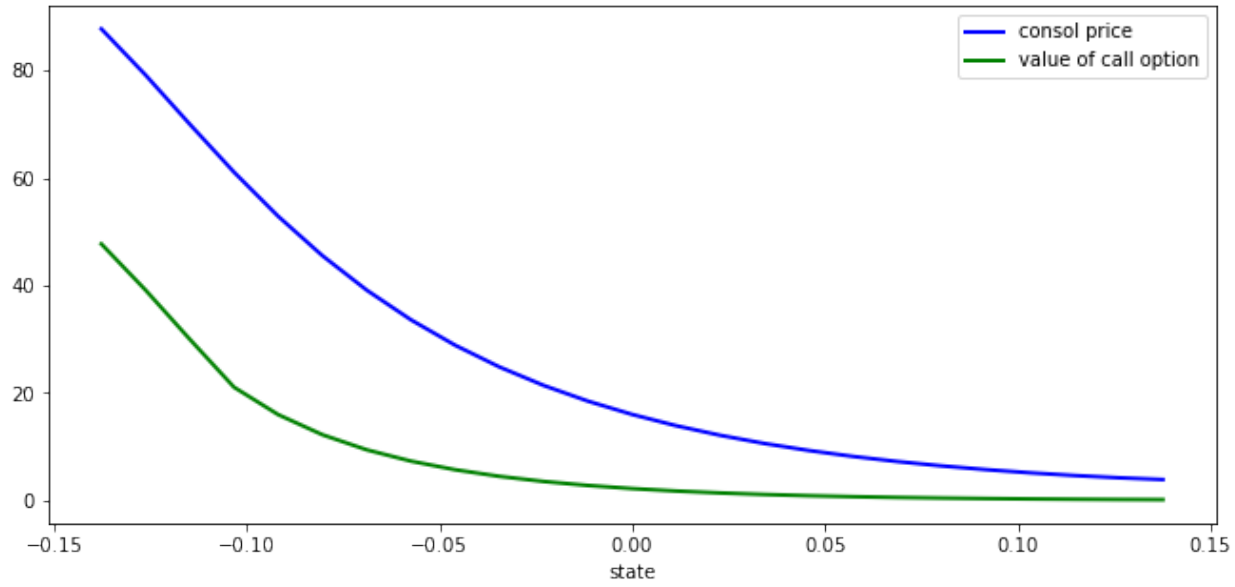
(continues on next page)

(continued from previous page)

```
        w = w_new

    return w
```

Here's a plot of w compared to the consol price when $P_S = 40$

```
ap = AssetPriceModel(β=0.9)
ζ = 1.0
strike_price = 40

x = ap.mc.state_values
p = consol_price(ap, ζ)
w = call_option(ap, ζ, strike_price)

fig, ax = plt.subplots()
ax.plot(x, p, 'b-', lw=2, label='consol price')
ax.plot(x, w, 'g-', lw=2, label='value of call option')
ax.set_xlabel("state")
ax.legend(loc='upper right')
plt.show()
```



In high values of the Markov growth state, the value of the option is close to zero.

This is despite the facts that the Markov chain is irreducible and that low states — where the consol prices are high — will be visited recurrently.

The reason for low valuations in high Markov growth states is that $\beta = 0.9$, so future payoffs are discounted substantially.

59.4.4 Risk-Free Rates

Let's look at risk-free interest rates over different periods.

The One-period Risk-free Interest Rate

As before, the stochastic discount factor is $m_{t+1} = \beta g_{t+1}^{-\gamma}$.

It follows that the reciprocal R_t^{-1} of the gross risk-free interest rate R_t in state x is

$$\mathbb{E}_t m_{t+1} = \beta \sum_{y \in S} P(x,y) g(y)^{-\gamma}$$

We can write this as

$$m_1 = \beta M \mathbb{1}$$

where the i-th element of m_1 is the reciprocal of the one-period gross risk-free interest rate in state x_i.

Other Terms

Let m_j be an $n \times 1$ vector whose i th component is the reciprocal of the j -period gross risk-free interest rate in state x_i.

Then $m_1 = \beta M$, and $m_{j+1} = M m_j$ for $j \geq 1$.

59.5 Exercises

59.5.1 Exercise 1

In the lecture, we considered **ex-dividend assets**.

A **cum-dividend** asset is a claim to the stream $d_t, d_{t+1}, \ldots$.

Following (1), find the risk-neutral asset pricing equation for one unit of a cum-dividend asset.

With a constant, non-random dividend stream $d_t = d > 0$, what is the equilibrium price of a cum-dividend asset?

With a growing, non-random dividend process $d_t = g d_t$ where $0 < g\beta < 1$, what is the equilibrium price of a cum-dividend asset?

59.5.2 Exercise 2

Consider the following primitives

```
n = 5
P = np.full((n, n), 0.0125)
P[range(n), range(n)] += 1 - P.sum(1)
# State values of the Markov chain
s = np.array([0.95, 0.975, 1.0, 1.025, 1.05])
γ = 2.0
β = 0.94
```

Let g be defined by $g(x) = x$ (that is, g is the identity map).

Compute the price of the Lucas tree.

Do the same for

- the price of the risk-free consol when $\zeta = 1$
- the call option on the consol when $\zeta = 1$ and $p_S = 150.0$

59.5.3 Exercise 3

Let's consider finite horizon call options, which are more common than the infinite horizon variety.

Finite horizon options obey functional equations closely related to (17).

A k period option expires after k periods.

If we view today as date zero, a k period option gives the owner the right to exercise the option to purchase the risk-free consol at the strike price p_S at dates $0, 1, \ldots, k-1$.

The option expires at time k.

Thus, for $k = 1, 2, \ldots$, let $w(x, k)$ be the value of a k-period option.

It obeys

$$w(x,k) = \max\left\{\beta \sum_{y \in S} P(x,y) g(y)^{-\gamma} w(y, k-1),\ p(x) - p_S\right\}$$

where $w(x, 0) = 0$ for all x.

We can express the preceding as the sequence of nonlinear vector equations

$$w_k = \max\{\beta M w_{k-1},\ p - p_S \mathbb{1}\} \quad k = 1, 2, \ldots \quad \text{with } w_0 = 0$$

Write a function that computes w_k for any given k.

Compute the value of the option with `k = 5` and `k = 25` using parameter values as in Exercise 1.

Is one higher than the other? Can you give intuition?

59.6 Solutions

59.6.1 Exercise 1

For a cum-dividend asset, the basic risk-neutral asset pricing equation is

$$p_t = d_t + \beta \mathbb{E}_t[p_{t+1}]$$

With constant dividends, the equilibrium price is

$$p_t = \frac{1}{1-\beta} d_t$$

With a growing, non-random dividend process, the equilibrium price is

$$p_t = \frac{1}{1-\beta g} d_t$$

59.6.2 Exercise 2

First, let's enter the parameters:

```
n = 5
P = np.full((n, n), 0.0125)
P[range(n), range(n)] += 1 - P.sum(1)
s = np.array([0.95, 0.975, 1.0, 1.025, 1.05])  # State values
mc = qe.MarkovChain(P, state_values=s)

γ = 2.0
β = 0.94
ζ = 1.0
p_s = 150.0
```

Next, we'll create an instance of `AssetPriceModel` to feed into the functions

```
apm = AssetPriceModel(β=β, mc=mc, γ=γ, g=lambda x: x)
```

Now we just need to call the relevant functions on the data:

```
tree_price(apm)
```

```
array([29.47401578, 21.93570661, 17.57142236, 14.72515002, 12.72221763])
```

```
consol_price(apm, ζ)
```

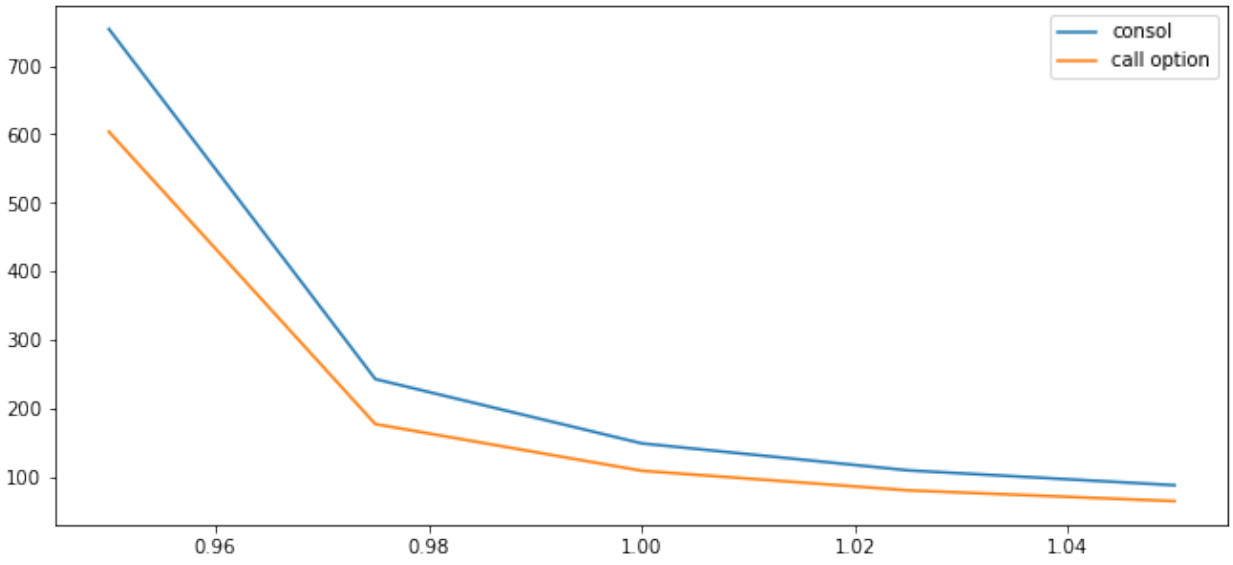
```
array([753.87100476, 242.55144082, 148.67554548, 109.25108965,
        87.56860139])
```

```
call_option(apm, ζ, p_s)
```

```
array([603.87100476, 176.8393343 , 108.67734499,  80.05179254,
        64.30843748])
```

Let's show the last two functions as a plot

```
fig, ax = plt.subplots()
ax.plot(s, consol_price(apm, ζ), label='consol')
ax.plot(s, call_option(apm, ζ, p_s), label='call option')
ax.legend()
plt.show()
```



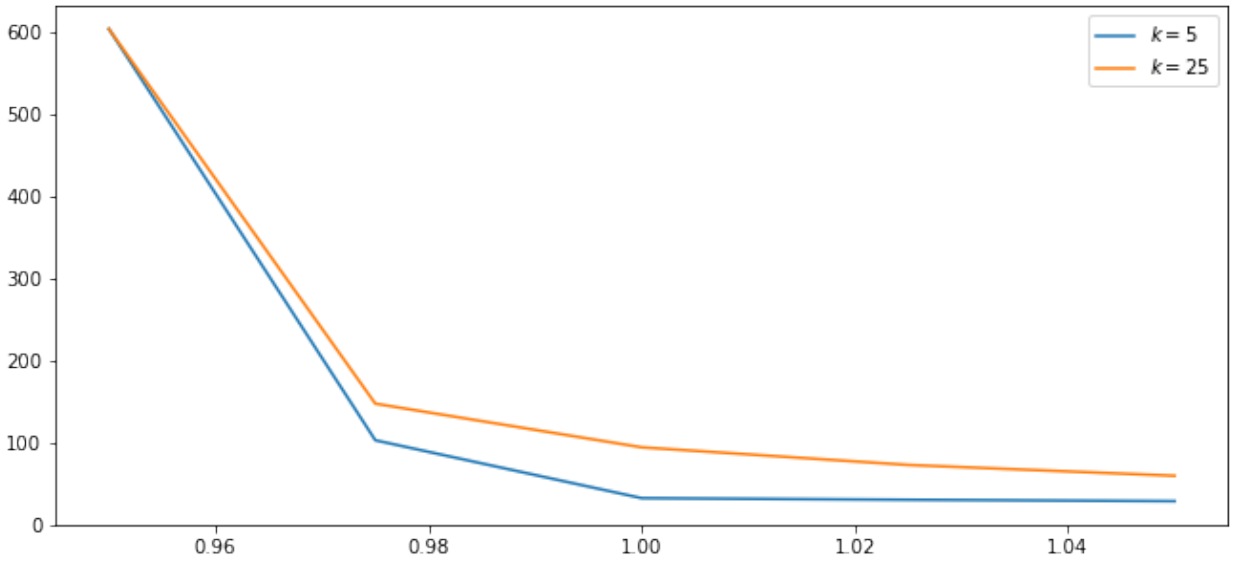
59.6.3 Exercise 3

Here's a suitable function:

```
def finite_horizon_call_option(ap, ζ, p_s, k):
    """
    Computes k period option value.
    """
    # Simplify names, set up matrices
    β, γ, P, y = ap.β, ap.γ, ap.mc.P, ap.mc.state_values
    M = P * ap.g(y)**(- γ)

    # Make sure that a unique solution exists
    ap.test_stability(M)


    # Compute option price
    p = consol_price(ap, ζ)
    w = np.zeros(ap.n)
    for i in range(k):
        # Maximize across columns
        w = np.maximum(β * M @ w, p - p_s)

    return w
```

Now let's compute the option values at `k=5` and `k=25`

```
fig, ax = plt.subplots()
for k in [5, 25]:
    w = finite_horizon_call_option(apm, ζ, p_s, k)
    ax.plot(s, w, label=rf'$k = {k}$')
ax.legend()
plt.show()
```



Not surprisingly, the option has greater value with larger k.

This is because the owner has a longer time horizon over which he or she may exercise the option.

CHAPTER

SIXTY

COMPETITIVE EQUILIBRIA WITH ARROW SECURITIES

60.1 Introduction

This lecture is a laboratory for experimenting with competitive equilibria of an infinite-horizon pure exchange economy with

- Markov endowments
- Complete markets in one-period Arrow state-contingent securities
- Discounted expected utility preferences of a kind often used in macroeconomics and finance
- Common expected utility preferences across agents
- Common beliefs across agents
- A constant relative risk aversion (CRRA) one-period utility function that implies the existence of a representative consumer whose consumption process can be plugged into a formula for the pricing kernel for one-step Arrow securities and thereby determine equilbrium prices before determining an equilibrium distribution of wealth
- Diverse endowments across agents that provide motivations to reallocate across time and Markov states

We impose restrictions that allow us to **Bellmanize** competitive equilibrium prices and quantities

We use Bellman equations to describe

- asset prices
- continuation wealths
- state-by-state natural debt limits

In the course of presenting the model we shall describe these important ideas

- a **resolvent operator** widely used in this class of models
- state-by-state **borrowing limits** required in infinite horizon economies
- absence of **borrowing limits** in finite horizon economies
- a counterpart of the law of iterated expectations known as a **law of iterated values**
- a **state-variable degeneracy** that prevails within a competitive equilibrium and that explains many appearances of resolvent operators

60.2 The setting

In effect, this lecture implements a Python version of the model presented in section 9.3.3 of Ljungqvist and Sargent [LS18].

60.2.1 Preferences and endowments

In each period $t \geq 0$, there is a realization of a stochastic event $s_t \in \mathbf{S}$.

Let the history of events up and until time t be denoted $s^t = [s_0, s_1, \ldots, s_{t-1}, s_t]$.

Sometimes we inadvertently reverse the recording order and denote a history as $s^t = [s_t, s_{t-1}, \ldots, s_1, s_0]$.

The unconditional probability of observing a particular sequence of events s^t is given by a probability measure $\pi_t(s^t)$.

For $t > \tau$, we write the probability of observing s^t conditional on the realization of s^τas $\pi_t(s^t|s^\tau)$.

We assume that trading occurs after observing s_0, which we capture by setting $\pi_0(s_0) = 1$ for the initially given value of s_0.

In this lecture we shall follow much of the literatures in macroeconomics and econometrics and assume that $\pi_t(s^t)$ is induced by a Markov process.

There are I consumers named $i = 1, \ldots, I$.

Consumer i owns a stochastic endowment of one good $y_t^i(s^t)$ that depends on the history s^t.

The history s^t is publicly observable.

Consumer i purchases a history-dependent consumption plan $c^i = \{c_t^i(s^t)\}_{t=0}^\infty$

Consumer i orders consumption plans by

$$U_i(c^i) = \sum_{t=0}^{\infty} \sum_{s^t} \beta^t u_i[c_t^i(s^t)]\pi_t(s^t),$$

where $0 < \beta < 1$.

The right side is equal to $E_0 \sum_{t=0}^\infty \beta^t u_i(c_t^i)$, where E_0 is the mathematical expectation operator, conditioned on s_0.

Here $u_i(c)$ is an increasing, twice continuously differentiable, strictly concave function of consumption $c \geq 0$ of one good.

The utility function satisfies the Inada condition

$$\lim_{c \downarrow 0} u_i'(c) = +\infty.$$

This condition implies that each agent chooses strictly positive consumption for every date-history pair.

Those interior solutions enable us to confine our analysis to Euler equations that hold with equality and also guarantee that **natural debt limits** don't bind in economies like ours with sequential trading of Arrow securities.

We adopt the assumption, routinely employed in much of macroeconomics, that consumers share probabilities $\pi_t(s^t)$ for all t and s^t.

A **feasible allocation** satisfies

$$\sum_i c_t^i(s^t) \leq \sum_i y_t^i(s^t)$$

for all t and for all s^t.

60.2.2 Recursive formulation

Following descriptions in section 9.3.3 of Ljungqvist and Sargent [LS18] chapter 9, we set up a competitive equilibrium of a pure exchange economy with complete markets in one-period Arrow securities.

When endowments $y^i(s)$ are all functions of a common Markov state s, the pricing kernel takes the form $Q(s'|s)$.

These enable us to provide a recursive formulation of a consumer's optimization problem.

Consumer i's state at time t is its financial wealth a_t^i and Markov state s_t.

Let $v^i(a, s)$ be the optimal value of consumer i's problem starting from state (a, s).

- $v^i(a, s)$ is the maximum expected discounted utility that consumer i with current financial wealth a can attain in state s.

The optimal value function satisfies the Bellman equation

$$v^i(a,s) = \max_{c,\hat{a}(s')} \left\{ u_i(c) + \beta \sum_{s'} v^i[\hat{a}(s'), s']\pi(s'|s) \right\}$$

where maximization is subject to the budget constraint

$$c + \sum_{s'} \hat{a}(s')Q(s'|s) \leq y^i(s) + a$$

and also the constraints

$$\begin{aligned} c &\geq 0, \\ -\hat{a}(s') &\leq \bar{A}^i(s'), \qquad \forall s'. \end{aligned}$$

with the second constraint evidently being a set of state-by-state debt limits.

Note that the value function and decision rule that solve the Bellman equation implicitly depend on the pricing kernel $Q(\cdot|\cdot)$ because it appears in the agent's budget constraint.

Use the first-order conditions for the problem on the right of the Bellman equation and a Benveniste-Scheinkman formula and rearrange to get

$$Q(s_{t+1}|s_t) = \frac{\beta u_i'(c_{t+1}^i)\pi(s_{t+1}|s_t)}{u_i'(c_t^i)},$$

where it is understood that $c_t^i = c^i(s_t)$ and $c_{t+1}^i = c^i(s_{t+1})$.

A **recursive competitive equilibrium** is an initial distribution of wealth $\vec{a}_0$, a set of borrowing limits $\{\bar{A}^i(s)\}_{i=1}^I$, a pricing kernel $Q(s'|s)$, sets of value functions $\{v^i(a,s)\}_{i=1}^I$, and decision rules $\{c^i(s), a^i(s)\}_{i=1}^I$ such that

- The state-by-state borrowing constraints satisfy the recursion
$$\bar{A}^i(s) = y^i(s) + \sum_{s'} Q(s'|s)\bar{A}^i(s')$$
- For all i, given a_0^i, $\bar{A}^i(s)$, and the pricing kernel, the value functions and decision rules solve the consumer's problem;
- For all realizations of $\{s_t\}_{t=0}^\infty$, the consumption and asset portfolios $\{\{c_t^i,\ \{\hat{a}_{t+1}^i(s')\}_{s'}\}_i\}_t$ satisfy $\sum_i c_t^i = \sum_i y^i(s_t)$ and $\sum_i \hat{a}_{t+1}^i(s') = 0$ for all t and s'.
- The initial financial wealth vector $\vec{a}_0$ satisfies $\sum_{i=1}^I a_0^i = 0$.

The third condition asserts that there are zero net aggregate claims in all Markov states.

The fourth condition asserts that the economy is closed and starts from a situation in which there are zero net claims in the aggregate.

If an allocation and prices in a recursive competitive equilibrium are to be consistent with the equilibrium allocation and price system that prevail in a corresponding complete markets economy with all trades occurring at time 0, we must impose that $a_0^i = 0$ for $i = 1, \ldots, I$.

That is what assures that at time 0 the present value of each agent's consumption equals the present value of his endowment stream, the single budget constraint in arrangement with all trades occurring at time 0.

Starting the system with $a_0^i = 0$ forall i has a striking implication that we can call **state variable degeneracy**.

Here is what we mean by **state variable degeneracy**:

Notice that although there are two state variables in the value function $v^i(a, s)$, within a recursive competitive equilibrium starting from $a_0^i = 0 \; \forall i$ at the starting Markov state s_0, two outcomes prevail:

- $a_0^i = 0$ for all i whenever the Markov state s_t returns to s_0.
- Financial wealth a is an exact function of the Markov state s.

The first finding asserts that each household recurrently visits the zero financial wealth state with which it began life.

The second finding asserts that the exogenous Markov state is all we require to track an individual within a competitive equilibrium.

Financial wealth turns out to be redundant because it is an exact function of the Markov state for each individual.

This outcome depends critically on there being complete markets in Arrow securities.

60.2.3 Markov asset prices primer

Let's start with a brief summary of formulas for computing asset prices in a Markov setting.

The setup assumes the following infrastructure

- Markov states: $s \in S = [\bar{s}_1, \ldots, \bar{s}_n]$ governed by an n-state Markov chain with transition probability
$$P_{ij} = \Pr\left\{s_{t+1} = \bar{s}_j \mid s_t = \bar{s}_i\right\}$$
- A collection $k = 1, \ldots, K$ of $n \times 1$ vectors of K assets that pay off $d^k(s)$ in state s
- An $n \times n$ matrix pricing kernel Q for one-period Arrow securities, where Q_{ij} = price at time t in state $s_t = \bar{s}_i$ of one unit of consumption when $s_{t+1} = \bar{s}_j$ at time $t+1$:
$$Q_{ij} = \text{Price}\left\{s_{t+1} = \bar{s}_j \mid s_t = \bar{s}_i\right\}$$
- The price of risk-free one-period bond in state i is $R_i^{-1} = \sum_j Q_{i,j}$
- The gross rate of return on a one-period risk-free bond Markov state $\bar{s}_i$ is $R_i = (\sum_j Q_{i,j})^{-1}$

At this point, we'll take the pricing kernel Q as exogenous, i.e., determined outside the model

Two examples would be

- $Q = \beta P$ where $\beta \in (0, 1)$
- $Q = SP$ where S is an $n \times n$ matrix of *stochastic discount factors*

We'll write down implications of Markov asset pricing in a nutshell for two types of assets

- the price in Markov state s at time t of a **cum dividend** stock that entitles the owner at the beginning of time t to the time t dividend and the option to sell the asset at time $t+1$. The price evidently satisfies $p^k(\bar{s}_i) = d^k(\bar{s}_i) + \sum_j Q_{ij} p^k(\bar{s}_j)$, which implies that the vector p^k satisfies $p^k = d^k + Qp^k$ which implies the formula

$$p^k = (I - Q)^{-1} d^k$$

- the price in Markov state s at time t of an **ex dividend** stock that entitles the owner at the end of time t to the time $t+1$ dividend and the option to sell the stock at time $t+1$. The price is

$$p^k = (I - Q)^{-1} Q d^k$$

Below, we describe an equilibrium model with trading of one-period Arrow securities in which the pricing kernel is endogenous.

In constructing our model, we'll repeatedly encounter formulas that remind us of our asset pricing formulas.

60.2.4 Multi-step-forward transition probabilities and pricing kernels

The (i, j) component of the k-step ahead transition probability P^k is

$$Prob(s_{t+k} = \bar{s}_j | s_t = \bar{s}_i) = P^k_{i,j}$$

The (i, j) component of the k-step ahead pricing kernel Q^k is

$$Q^{(k)}(s_{t+k} = \bar{s}_j | s_t = \bar{s}_i) = Q^k_{i,j}$$

We'll use these objects to state a useful property in asset pricing theory.

60.2.5 Laws of iterated expectations and iterated values

A **law of iterated values** has a mathematical structure that parallels the **law of iterated expectations**

We can describe its structure readily in the Markov setting of this lecture

Recall the following recursion satisfied j step ahead transition probabilites for our finite state Markov chain:

$$P_j(s_{t+j}|s_t) = \sum_{s_{t+1}} P_{j-1}(s_{t+j}|s_{t+1}) P(s_{t+1}|s_t)$$

We can use this recursion to verify the law of iterated expectations applied to computing the conditional expectation of a random variable $d(s_{t+j})$ conditioned on s_t via the following string of equalities

$$\begin{aligned} E\left[Ed(s_{t+j})|s_{t+1}\right]|s_t &= \sum_{s_{t+1}} \left[\sum_{s_{t+j}} d(s_{t+j}) P_{j-1}(s_{t+j}|s_{t+1})\right] P(s_{t+1}|s_t) \\ &= \sum_{s_{t+j}} d(s_{t+j}) \left[\sum_{s_{t+1}} P_{j-1}(s_{t+j}|s_{t+1}) P(s_{t+1}|s_t)\right] \\ &= \sum_{s_{t+j}} d(s_{t+j}) P_j(s_{t+j}|s_t) \\ &= Ed(s_{t+j})|s_t \end{aligned}$$

The pricing kernel for j step ahead Arrow securities satisfies the recursion

$$Q_j(s_{t+j}|s_t) = \sum_{s_{t+1}} Q_{j-1}(s_{t+j}|s_{t+1}) Q(s_{t+1}|s_t)$$

The time t **value** in Markov state s_t of a time $t+j$ payout $d(s_{t+j})$ is

$$V(d(s_{t+j})|s_t) = \sum_{s_{t+j}} d(s_{t+j}) Q_j(s_{t+j}|s_t)$$

The **law of iterated values** states

$$V\left[V(d(s_{t+j})|s_{t+1})\right]|s_t = V(d(s_{t+j}))|s_t$$

We verify it by pursuing the following a string of inequalities that are counterparts to those we used to verify the law of iterated expectations:

$$\begin{aligned} V\left[V(d(s_{t+j})|s_{t+1})\right]|s_t &= \sum_{s_{t+1}} \left[\sum_{s_{t+j}} d(s_{t+j}) Q_{j-1}(s_{t+j}|s_{t+1})\right] Q(s_{t+1}|s_t) \\ &= \sum_{s_{t+j}} d(s_{t+j}) \left[\sum_{s_{t+1}} Q_{j-1}(s_{t+j}|s_{t+1}) Q(s_{t+1}|s_t)\right] \\ &= \sum_{s_{t+j}} d(s_{t+j}) Q_j(s_{t+j}|s_t) \\ &= EV(d(s_{t+j}))|s_t \end{aligned}$$

60.3 General equilibrium model (pure exchange)

Now we are ready to do some fun calculations.

We find it interesting to think in terms of analytical **inputs** into and **outputs** from our general equilibrium theorizing.

60.3.1 Inputs

- Markov states: $s \in S = [\bar{s}_1, \ldots, \bar{s}_n]$ governed by an n-state Markov chain with transition probability
$$P_{ij} = \Pr\left\{s_{t+1} = \bar{s}_j \mid s_t = \bar{s}_i\right\}$$
- A collection of $K \times 1$ vectors of individual k endowments: $y^k(s), k = 1, \ldots, K$
- An $n \times 1$ vector of aggregate endowment: $y(s) \equiv \sum_{k=1}^K y^k(s)$
- A collection of $K \times 1$ vectors of individual k consumptions: $c^k(s), k = 1, \ldots, K$
- A collection of restrictions on feasible consumption allocations for $s \in S$:
$$c(s) = \sum_{k=1}^K c^k(s) \leq y(s)$$
- Preferences: a common utility functional across agents $E_0 \sum_{t=0}^\infty \beta^t u(c_t^k)$ with CRRA one-period utility function $u(c)$ and discount factor $\beta \in (0,1)$

The one-period utility function is

$$u(c) = \frac{c^{1-\gamma}}{1-\gamma}$$

so that

$$u'(c) = c^{-\gamma}$$

60.3.2 Outputs

- An $n \times n$ matrix pricing kernel Q for one-period Arrow securities, where Q_{ij} = price at time t in state $s_t \bar{s}_i$ of one unit of consumption when $s_{t+1} = \bar{s}_j$ at time $t+1$
- pure exchange so that $c(s) = y(s)$
- an $K \times 1$ vector distribution of wealth vector α, $\alpha_k \geq 0, \sum_{k=1}^{K} \alpha_k = 1$
- A collection of $n \times 1$ vectors of individual k consumptions: $c^k(s), k = 1, \ldots, K$

60.3.3 Matrix Q to represent pricing kernel

For any agent $k \in [1, \ldots, K]$, at the equilibrium allocation, the one-period Arrow securities pricing kernel satisfies

$$Q_{ij} = \beta \left(\frac{c^k(\bar{s}_j)}{c^k(\bar{s}_i)} \right)^{-\gamma} P_{ij}$$

where Q is an $n \times n$ matrix

This follows from agent k's first-order necessary conditions.

But with the CRRA preferences that we have assumed, individual consumptions vary proportionately with aggregate consumption and therefore with the aggregate endowment.

- This is a consequence of our preference specification implying that **Engle curves** affine in wealth and therefore satisfy conditions for **Gorman aggregation**

Thus,

$$c^k(s) = \alpha_k c(s) = \alpha_k y(s)$$

for an arbitrary **distribution of wealth** in the form of an $K \times 1$ vector α that satisfies

$$\alpha_k \in (0,1), \quad \sum_{k=1}^{K} \alpha_k = 1$$

This means that we can compute the pricing kernel from

$$Q_{ij} = \beta \left(\frac{y_j}{y_i} \right)^{-\gamma} P_{ij}$$

Note that Q_{ij} is independent of vector α.

Thus, we have the

Key finding: We can compute competitive equilibrium **prices** prior to computing a **distribution of wealth**.

60.3.4 Values

Having computed an equilibrium pricing kernel Q, we can compute several **values** that are required to pose or represent the solution of an individual household's optimum problem.

We denote an $K \times 1$ vector of state-dependent values of agents' endowments in Markov state s as

$$A(s) = \begin{bmatrix} A^1(s) \\ \vdots \\ A^K(s) \end{bmatrix}, \quad s \in [\bar{s}_1, \ldots, \bar{s}_n]$$

and an $n \times 1$ vector of continuation endowment values for each individual k as

$$A^k = \begin{bmatrix} A^k(\bar{s}_1) \\ \vdots \\ A^k(\bar{s}_n) \end{bmatrix}, \quad k \in [1, \ldots, K]$$

A^k of consumer i satisfies

$$A^k = [I - Q]^{-1} [y^k]$$

where

$$y^k = \begin{bmatrix} y^k(\bar{s}_1) \\ \vdots \\ y^k(\bar{s}_n) \end{bmatrix} \equiv \begin{bmatrix} y_1^k \\ \vdots \\ v_n^k \end{bmatrix}$$

In a competitive equilibrium of an **infinite horizon** economy with sequential trading of one-period Arrow securities, $A^k(s)$ serves as a state-by-state vector of **debt limits** on the quantities of one-period Arrow securities paying off in state s at time $t+1$ that individual k can issue at time t.

These are often called **natural debt limits**.

Evidently, they equal the maximum amount that it is feasible for individual i to repay even if he consumes zero goods forevermore.

Remark: If we have an Inada condition at zero consumption or just impose that consumption be nonnegative, then in a **finite horizon** economy with sequential trading of one-period Arrow securities there is no need to impose natural debt limits.

60.3.5 Continuation wealths

Continuation wealths play an important role in Bellmanizing a competitive equilibrium with sequential trading of a complete set of one-period Arrow securities.

We denote an $K \times 1$ vector of state-dependent continuation wealths in Markov state s as

$$\psi(s) = \begin{bmatrix} \psi^1(s) \\ \vdots \\ \psi^K(s) \end{bmatrix}, \quad s \in [\bar{s}_1, \ldots, \bar{s}_n]$$

and an $n \times 1$ vector of continuation wealths for each individual i as

$$\psi^k = \begin{bmatrix} \psi^k(\bar{s}_1) \\ \vdots \\ \psi^k(\bar{s}_n) \end{bmatrix}, \quad k \in [1, \ldots, K]$$

Continuation wealth ψ^k of consumer i satisfies

$$\psi^k = [I - Q]^{-1} [\alpha_k y - y^k]$$

where

$$y^k = \begin{bmatrix} y^k(\bar{s}_1) \\ \vdots \\ y^k(\bar{s}_n) \end{bmatrix}, \quad y = \begin{bmatrix} y(\bar{s}_1) \\ \vdots \\ y(\bar{s}_n) \end{bmatrix}$$

Note that $\sum_{k=1}^K \psi^k = 0_{n \times 1}$.

Remark: At the initial state $s_0 \in [\bar{s}_1, \ldots, \bar{s}_n]$ the continuation wealth $\psi^k(s_0) = 0$ for all agents $k = 1, \ldots, K$. This indicates that the economy begins with all agents being debt-free and financial-asset-free at time 0, state s_0.

Remark: Note that all agents' continuation wealths recurrently return to zero when the Markov state returns to whatever value s_0 it had at time 0.

60.3.6 Optimal portfolios

A nifty feature of the model is that optimal portfolios for a type k agent equal the continuation wealths that we have just computed.

Thus, agent k's state-by-state purchases of Arrow securities next period depend only on next period's Markov state and equal

$$a_k(s) = \psi^k(s), \quad s \in [\bar{s}_1, \ldots, \bar{s}_n]$$

60.3.7 Equilibrium wealth distribution α

With the initial state being a particular state $s_0 \in [\bar{s}_1, \ldots, \bar{s}_n]$, we must have

$$\psi^k(s_0) = 0, \quad k = 1, \ldots, K$$

which means the equilibrium distribution of wealth satisfies

$$\alpha_k = \frac{V_z y^k}{V_z y}$$

where $V \equiv [I - Q]^{-1}$ and z is the row index corresponding to the initial state s_0.

Since $\sum_{k=1}^K V_z y^k = V_z y$, $\sum_{k=1}^K \alpha_k = 1$.

In summary, here is the logical flow of an algorithm to compute a competitive equilibrium:

- compute Q from the aggregate allocation and the above formula
- compute the distribution of wealth α from the formula just given
- Using α assign each consumer k the share α_k of the aggregate endowment at each state
- return to the α-dependent formula for continuation wealths and compute continuation wealths
- equate agent k's portfolio to its continuation wealth state by state

We can also add formulas for optimal value functions in a competitive equilibrium with trades in a complete set of one-period state-contingent Arrow securities.

Call the optimal value functions J^k for consumer k.

For the infinite horizon economy now under study, the formula is

$$J^k = (I - \beta P)^{-1} u(\alpha_k y), \quad u(c) = \frac{c^{1-\gamma}}{1-\gamma}$$

where it is understood that $u(\alpha_k y)$ is a vector.

We are ready to dive into some Python code.

As usual, we start with Python imports.

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

```
np.set_printoptions(suppress=True)
```

First, we create a Python class to compute the objects that comprise a competitive equilibrium with sequential trading of one-period Arrow securities.

The reader will notice that the code is set up to handle finite-horizon economies indexed by horizon T.

We'll study some finite horizon economies after we look at some infinite-horizon economies.

```
class RecurCompetitive:
    """
    A class that represents a recursive competitive economy
    with one-period Arrow securities.
    """

    def __init__(self,
                 s,         # state vector
                 P,         # transition matrix
                 ys,        # endowments ys = [y1, y2, .., yI]
                 γ=0.5,     # risk aversion
                 β=0.98,    # discount rate
                 T=None):   # time horizon, none if infinite

        # preference parameters
        self.γ = γ
        self.β = β

        # variables dependent on state
        self.s = s
        self.P = P
        self.ys = ys
        self.y = np.sum(ys, 1)

        # dimensions
        self.n, self.K = ys.shape

        # compute pricing kernel
        self.Q = self.pricing_kernel()

        # compute price of risk-free one-period bond
        self.PRF = self.price_risk_free_bond()

        # compute risk-free rate
        self.R = self.risk_free_rate()

        # V = [I - Q]^{-1} (infinite case)
        if T is None:
            self.T = None
            self.V = np.empty((1, n, n))
            self.V[0] = np.linalg.inv(np.eye(n) - self.Q)
        # V = [I + Q + Q^2 + ... + Q^T] (finite case)
        else:
            self.T = T
            self.V = np.empty((T+1, n, n))
            self.V[0] = np.eye(n)

            Qt = np.eye(n)
            for t in range(1, T+1):
                Qt = Qt.dot(self.Q)
                self.V[t] = self.V[t-1] + Qt
```

(continues on next page)

(continued from previous page)

```
        # natural debt limit
        self.A = self.V[-1] @ ys

    def u(self, c):
        "The CRRA utility"

        return c ** (1 - self.γ) / (1 - self.γ)

    def u_prime(self, c):
        "The first derivative of CRRA utility"

        return c ** (-self.γ)

    def pricing_kernel(self):
        "Compute the pricing kernel matrix Q"

        c = self.y

        n = self.n
        Q = np.empty((n, n))

        for i in range(n):
            for j in range(n):
                ratio = self.u_prime(c[j]) / self.u_prime(c[i])
                Q[i, j] = self.β * ratio * P[i, j]

        self.Q = Q

        return Q

    def wealth_distribution(self, s0_idx):
        "Solve for wealth distribution α"

        # set initial state
        self.s0_idx = s0_idx

        # simplify notations
        n = self.n
        Q = self.Q
        y, ys = self.y, self.ys

        # row of V corresponding to s0
        Vs0 = self.V[-1, s0_idx, :]
        α = Vs0 @ self.ys / (Vs0 @ self.y)

        self.α = α

        return α

    def continuation_wealths(self):
        "Given α, compute the continuation wealths ψ"

        diff = np.empty((n, K))
        for k in range(K):
            diff[:, k] = self.α[k] * self.y - self.ys[:, k]

        ψ = self.V @ diff
```

(continues on next page)

(continued from previous page)

```
        self.ψ = ψ

        return ψ

    def price_risk_free_bond(self):
        "Give Q, compute price of one-period risk free bond"

        PRF = np.sum(self.Q, 0)
        self.PRF = PRF

        return PRF

    def risk_free_rate(self):
        "Given Q, compute one-period gross risk-free interest rate R"

        R = np.sum(self.Q, 0)
        R = np.reciprocal(R)
        self.R = R

        return R

    def value_functionss(self):
        "Given α, compute the optimal value functions J in equilibrium"

        n, T = self.n, self.T
        β = self.β
        P = self.P

        # compute (I - βP)^(-1) in infinite case
        if T is None:
            P_seq = np.empty((1, n, n))
            P_seq[0] = np.linalg.inv(np.eye(n) - β * P)
        # and (I + βP + ... + β^T P^T) in finite case
        else:
            P_seq = np.empty((T+1, n, n))
            P_seq[0] = np.eye(n)

            Pt = np.eye(n)
            for t in range(1, T+1):
                Pt = Pt.dot(P)
                P_seq[t] = P_seq[t-1] + Pt * β ** t

        # compute the matrix [u(a_1 y), ..., u(a_K, y)]
        flow = np.empty((n, K))
        for k in range(K):
            flow[:, k] = self.u(self.α[k] * self.y)

        J = P_seq @ flow

        self.J = J

        return J
```

60.3.8 Example 1

Please read the preceding class for default parameter values and the following Python code for the fundamentals of the economy.

Here goes.

```
# dimensions
K, n = 2, 2

# states
s = np.array([0, 1])

# transition
P = np.array([[.5, .5], [.5, .5]])

# endowments
ys = np.empty((n, K))
ys[:, 0] = 1 - s        # y1
ys[:, 1] = s            # y2
```

```
ex1 = RecurCompetitive(s, P, ys)
```

```
# endowments
ex1.ys
```

```
array([[1., 0.],
       [0., 1.]])
```

```
# pricing kernal
ex1.Q
```

```
array([[0.49, 0.49],
       [0.49, 0.49]])
```

```
# Risk free rate R
ex1.R
```

```
array([1.02040816, 1.02040816])
```

```
# natural debt limit, A = [A1, A2, ..., AI]
ex1.A
```

```
array([[25.5, 24.5],
       [24.5, 25.5]])
```

```
# when the initial state is state 1
print(f'α = {ex1.wealth_distribution(s0_idx=0)}')
print(f'ψ = \n{ex1.continuation_wealths()}')
print(f'J = \n{ex1.value_functionss()}')
```

```
α = [0.51 0.49]
ψ =
[[ 0. -0.]
```

(continues on next page)

(continued from previous page)

```
  [ 1. -1.]]]
J =
[[[71.41428429 70.        ]
  [71.41428429 70.        ]]]
```

```
# when the initial state is state 2
print(f'α = {ex1.wealth_distribution(s0_idx=1)}')
print(f'ψ = \n{ex1.continuation_wealths()}')
print(f'J = \n{ex1.value_functionss()}')
```

```
α = [0.49 0.51]
ψ =
[[[-1.  1.]
  [ 0. -0.]]]
J =
[[[70.         71.41428429]
  [70.         71.41428429]]]
```

60.3.9 Example 2

```
# dimensions
K, n = 2, 2

# states
s = np.array([1, 2])

# transition
P = np.array([[.5, .5], [.5, .5]])

# endowments
ys = np.empty((n, K))
ys[:, 0] = 1.5          # y1
ys[:, 1] = s            # y2
```

```
ex2 = RecurCompetitive(s, P, ys)
```

```
# endowments

print("ys = \n", ex2.ys)

# pricing kernal
print ("Q = \n", ex2.Q)

# Risk free rate R
print("R = ", ex2.R)
```

```
ys =
 [[1.5 1. ]
 [1.5 2. ]]
Q =
 [[0.49       0.41412558]
 [0.57977582 0.49      ]]
R =  [0.93477529 1.10604104]
```

```
# pricing kernal
ex2.Q
```

```
array([[0.49      , 0.41412558],
       [0.57977582, 0.49      ]])
```

```
# Risk free rate R
ex2.R
```

```
array([0.93477529, 1.10604104])
```

```
# natural debt limit, A = [A1, A2, ..., AI]
ex2.A
```

```
array([[69.30941886, 66.91255848],
       [81.73318641, 79.98879094]])
```

```
# when the initial state is state 1
print(f'α = {ex2.wealth_distribution(s0_idx=0)}')
print(f'ψ = \n{ex2.continuation_wealths()}')
print(f'J = \n{ex2.value_functionss()}')
```

```
α = [0.50879763 0.49120237]
ψ =
[[-0.         -0.        ]
  [ 0.55057195 -0.55057195]]]
J =
[[[122.907875   120.76397493]
  [123.32114686 121.17003803]]]
```

```
# when the initial state is state 1
print(f'α = {ex2.wealth_distribution(s0_idx=1)}')
print(f'ψ = \n{ex2.continuation_wealths()}')
print(f'J = \n{ex2.value_functionss()}')
```

```
α = [0.50539319 0.49460681]
ψ =
[[-0.46375886  0.46375886]
  [ 0.         -0.        ]]]
J =
[[[122.49598809 121.18174895]
  [122.907875   121.58921679]]]
```

60.3.10 Example 3

```
# dimensions
K, n = 2, 2

# states
s = np.array([1, 2])

# transition
λ = 0.9
P = np.array([[1-λ, λ], [0, 1]])

# endowments
ys = np.empty((n, K))
ys[:, 0] = [1, 0]          # y1
ys[:, 1] = [0, 1]          # y2
```

```
ex3 = RecurCompetitive(s, P, ys)
```

```
# endowments

print("ys = ", ex3.ys)

# pricing kernel
print ("Q = ", ex3.Q)

# Risk free rate R
print("R = ", ex3.R)
```

```
ys =  [[1. 0.]
 [0. 1.]]
Q =  [[0.098 0.882]
 [0.    0.98 ]]
R =  [10.20408163  0.53705693]
```

```
# pricing kernel
ex3.Q
```

```
array([[0.098, 0.882],
       [0.   , 0.98 ]])
```

```
# natural debt limit, A = [A1, A2, ..., AI]
ex3.A
```

```
array([[ 1.10864745, 48.89135255],
       [ 0.        , 50.        ]])
```

Note that the natural debt limit for agent 1 in state 2 is 0.

```
# when the initial state is state 1
print(f'α = {ex3.wealth_distribution(s0_idx=0)}')
print(f'ψ = \n{ex3.continuation_wealths()}')
print(f'J = \n{ex3.value_functionss()}')
```

```
α = [0.02217295 0.97782705]
ψ =
[[[ 0.          -0.        ]
  [ 1.10864745 -1.10864745]]]
J =
[[[14.89058394 98.88513796]
  [14.89058394 98.88513796]]]
```

```
# when the initial state is state 1
print(f'α = {ex3.wealth_distribution(s0_idx=1)}')
print(f'ψ = \n{ex3.continuation_wealths()}')
print(f'J = \n{ex3.value_functionss()}')
```

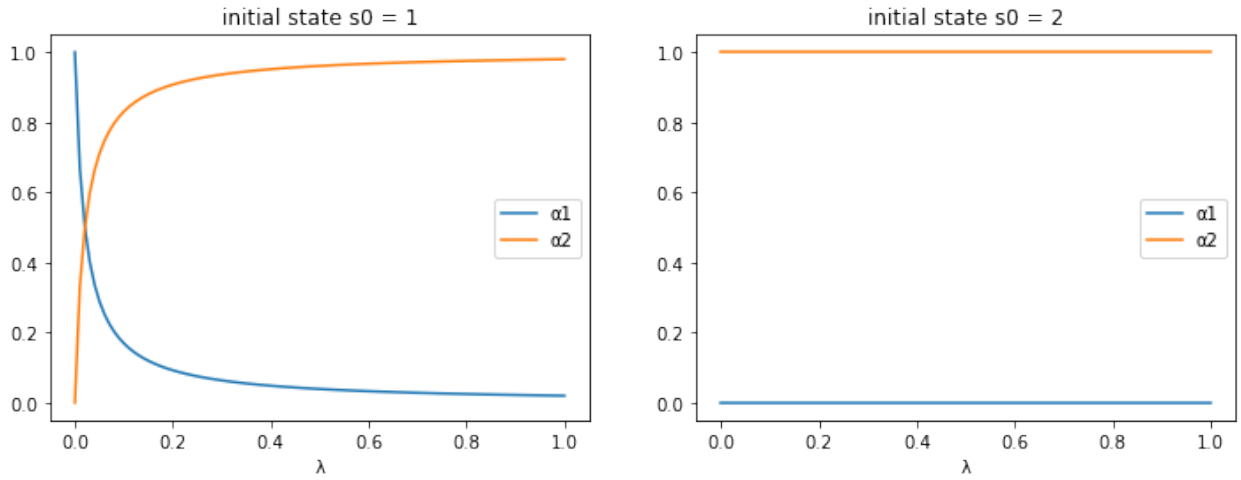
```
α = [0. 1.]
ψ =
[[[-1.10864745  1.10864745]
  [ 0.          0.        ]]]
J =
[[[  0. 100.]
  [  0. 100.]]]
```

For the specification of the Markov chain in example 3, let's take a look at how the equilibrium allocation changes as a function of transition probability λ.

```
λ_seq = np.linspace(0, 1, 100)

# prepare containers
αs0_seq = np.empty((len(λ_seq), 2))
αs1_seq = np.empty((len(λ_seq), 2))

for i, λ in enumerate(λ_seq):
    P = np.array([[1-λ, λ], [0, 1]])
    ex3 = RecurCompetitive(s, P, ys)

    # initial state s0 = 1
    α = ex3.wealth_distribution(s0_idx=0)
    αs0_seq[i, :] = α

    # initial state s0 = 2
    α = ex3.wealth_distribution(s0_idx=1)
    αs1_seq[i, :] = α
```

```
<ipython-input-3-222e6cc36b2b>:126: RuntimeWarning: divide by zero encountered in␣
↪reciprocal
  R = np.reciprocal(R)
```

```
fig, axs = plt.subplots(1, 2, figsize=(12, 4))

for i, αs_seq in enumerate([αs0_seq, αs1_seq]):
    for j in range(2):
        axs[i].plot(λ_seq, αs_seq[:, j], label=f'α{j+1}')
        axs[i].set_xlabel('λ')
        axs[i].set_title(f'initial state s0 = {s[i]}')
        axs[i].legend()

plt.show()
```



60.4 Example 4

```
# dimensions
K, n = 2, 3

# states
s = np.array([1, 2, 3])

# transition
λ = .9
μ = .9
δ = .05

P = np.array([[1-λ, λ, 0], [μ/2, μ, μ/2], [(1-δ)/2, (1-δ)/2, δ]])

# endowments
ys = np.empty((n, K))
ys[:, 0] = [.25, .75, .2]        # y1
ys[:, 1] = [1.25, .25, .2]       # y2
```

```
ex4 = RecurCompetitive(s, P, ys)
```

```
# endowments
print("ys = \n", ex4.ys)

# pricing kernal
print ("Q = \n", ex4.Q)

# Risk free rate R
print("R = ", ex4.R)

# natural debt limit, A = [A1, A2, ..., AI]
print("A = \n", ex4.A)

print('')

for i in range(1, 4):
```

(continues on next page)

(continued from previous page)

```
    # when the initial state is state i
    print(f"when the initial state is state {i}")
    print(f'α = {ex4.wealth_distribution(s0_idx=i-1)}')
    print(f'ψ = \n{ex4.continuation_wealths()}')
    print(f'J = \n{ex4.value_functionss()}\n')
```

```
ys =
 [[0.25 1.25]
 [0.75 0.25]
 [0.2  0.2 ]]
Q =
 [[0.098      1.08022498 0.        ]
 [0.36007499 0.882      0.69728222]
 [0.24038317 0.29440805 0.049     ]]
R =  [1.43172499 0.44313807 1.33997564]
A =
 [[-1.4141307  -0.45854174]
 [-1.4122483  -1.54005386]
 [-0.58434331 -0.3823659 ]]

when the initial state is state 1
α = [0.75514045 0.24485955]
ψ =
[[[ 0.          0.        ]
  [-0.81715447  0.81715447]
  [-0.14565791  0.14565791]]]
J =
[[[-2.65741909 -1.51322919]
  [-5.13103133 -2.92179221]
  [-2.65649938 -1.51270548]]]

when the initial state is state 2
α = [0.47835493 0.52164507]
ψ =
[[[ 0.5183286  -0.5183286 ]
  [ 0.         -0.        ]
  [ 0.12191319 -0.12191319]]]
J =
[[[-2.11505328 -2.20868477]
  [-4.08381377 -4.26460049]
  [-2.11432128 -2.20792037]]]

when the initial state is state 3
α = [0.60446648 0.39553352]
ψ =
[[[ 0.28216299 -0.28216299]
  [-0.37231938  0.37231938]
  [ 0.         -0.        ]]]
J =
[[[-2.37756442 -1.92325926]
  [-4.59067883 -3.71349163]
  [-2.37674158 -1.92259365]]]
```

60.5 Finite horizon economies

The Python class **RecurCompetitive** provided above also can be used to compute competitive equilibrium allocations and Arrow securities prices for finite horizon economies.

The setting is a finite-horizon version of the one above except that time now runs for $T+1$ periods $t \in \mathbf{T} = \{0, 1, \dots, T\}$.

Consequently, we want $T+1$ counterparts to objects described above, with one important exception: we won't need **borrowing limits** because they aren't required for a finite horizon economy in which a one-period utility function $u(c)$ satisfies an Inada condition that sets the marginal utility of consumption at zero consumption to zero. Nonnegativity of consumption choices at all $t \in \mathbf{T}$ automatically limits borrowing.

60.5.1 Continuation wealths

We denote an $K \times 1$ vector of state-dependent continuation wealths in Markov state s at time t as

$$\psi_t(s) = \begin{bmatrix} \psi^1(s) \\ \vdots \\ \psi^K(s) \end{bmatrix}, \quad s \in [\bar{s}_1, \dots, \bar{s}_n]$$

and an $n \times 1$ vector of continuation wealths for each individual k as

$$\psi_t^k = \begin{bmatrix} \psi_t^k(\bar{s}_1) \\ \vdots \\ \psi_t^k(\bar{s}_n) \end{bmatrix}, \quad k \in [1, \dots, K]$$

Continuation wealths ψ^k of consumer k satisfy

$$\begin{aligned} \psi_T^k &= [\alpha_k y - y^k] \\ \psi_{T-1}^k &= [I + Q][\alpha_k y - y^k] \\ \vdots &\qquad \vdots \\ \psi_0^k &= [I + Q + Q^2 + \cdots + Q^T][\alpha_k y - y^k] \end{aligned}$$

where

$$y^k = \begin{bmatrix} y^k(\bar{s}_1) \\ \vdots \\ y^k(\bar{s}_n) \end{bmatrix}, \quad y = \begin{bmatrix} y(\bar{s}_1) \\ \vdots \\ y(\bar{s}_n) \end{bmatrix}$$

Note that $\sum_{k=1}^K \psi_t^k = 0_{n \times 1}$ for all $t \in \mathbf{T}$.

Remark: At the initial state $s_0 \in [\bar{s}_1, \dots, \bar{s}_n]$, for all agents $k = 1, \dots, K$, continuation wealth $\psi_0^k(s_0) = 0$. This indicates that the economy begins with all agents being debt-free and financial-asset-free at time 0, state s_0.

Remark: Note that all agents' continuation wealths return to zero when the Markov state returns to whatever value s_0 it had at time 0. This will recur if the Markov chain makes the initial state s_0 recurrent.

With the initial state being a particular state $s_0 \in [\bar{s}_1, \dots, \bar{s}_n]$, we must have

$$\psi_0^k(s_0) = 0, \quad k = 1, \dots, K$$

which means the equilibrium distribution of wealth satisfies

$$\alpha_k = \frac{V_z y^k}{V_z y}$$

where now in our finite-horizon economy $V = [I + Q + Q^2 + \cdots + Q^T]$ and z is the row index corresponding to the initial state s_0.

Since $\sum_{k=1}^{K} V_z y^k = V_z y$, $\sum_{k=1}^{K} \alpha_k = 1$.

In summary, here is the logical flow of an algorithm to compute a competitive equilibrium with Arrow securities in our finite-horizon Markov economy:

- compute Q from the aggregate allocation and the above formula
- compute the distribution of wealth α from the formula just given
- Using α assign each consumer k the share α_k of the aggregate endowment at each state
- return to the α-dependent formula for continuation wealths and compute continuation wealths
- equate agent k's portfolio to its continuation wealth state by state

While for the infinite horizon economy, the formula for value functions is

$$J^k = (I - \beta P)^{-1} u(\alpha_k y), \quad u(c) = \frac{c^{1-\gamma}}{1-\gamma}$$

for the finite horizon economy the formula is

$$J_0^k = (I + \beta P + \cdots + \beta^T P^T) u(\alpha_k y)$$

where it is understood that $u(\alpha_k y)$ is a vector.

60.5.2 Finite horizon example

Below we revisit the economy defined in example 1, but set the time horizon to be $T = 10$.

```
# dimensions
K, n = 2, 2

# states
s = np.array([0, 1])

# transition
P = np.array([[.5, .5], [.5, .5]])

# endowments
ys = np.empty((n, K))
ys[:, 0] = 1 - s        # y1
ys[:, 1] = s            # y2
```

```
ex1_finite = RecurCompetitive(s, P, ys, T=10)
```

```
# (I + Q + Q^2 + ... + Q^T)
ex1_finite.V[-1]
```

```
array([[5.48171623, 4.48171623],
       [4.48171623, 5.48171623]])
```

```
# endowments
ex1_finite.ys
```

```
array([[1., 0.],
       [0., 1.]])
```

```
# pricing kernal
ex1_finite.Q
```

```
array([[0.49, 0.49],
       [0.49, 0.49]])
```

```
# Risk free rate R
ex1_finite.R
```

```
array([1.02040816, 1.02040816])
```

In the finite time horizon case, ψ and $\mathcal{J}$ are returned as sequences.

Components are ordered from $t = T$ to $t = 0$.

```
# when the initial state is state 2
print(f'α = {ex1_finite.wealth_distribution(s0_idx=0)}')
print(f'ψ = \n{ex1_finite.continuation_wealths()}\n')
print(f'J = \n{ex1_finite.value_functionss()}')
```

```
α = [0.55018351 0.44981649]
ψ =
[[[-0.44981649  0.44981649]
  [ 0.55018351 -0.55018351]]

 [[-0.40063665  0.40063665]
  [ 0.59936335 -0.59936335]]

 [[-0.35244041  0.35244041]
  [ 0.64755959 -0.64755959]]

 [[-0.30520809  0.30520809]
  [ 0.69479191 -0.69479191]]

 [[-0.25892042  0.25892042]
  [ 0.74107958 -0.74107958]]

 [[-0.21355851  0.21355851]
  [ 0.78644149 -0.78644149]]

 [[-0.16910383  0.16910383]
  [ 0.83089617 -0.83089617]]

 [[-0.12553824  0.12553824]
  [ 0.87446176 -0.87446176]]

 [[-0.08284397  0.08284397]
  [ 0.91715603 -0.91715603]]

 [[-0.04100358  0.04100358]
  [ 0.95899642 -0.95899642]]

 [[-0.         -0.        ]
```

(continues on next page)

(continued from previous page)

```
  [ 1.          -1.        ]]]

J =
[[[ 1.48348712  1.3413672 ]
  [ 1.48348712  1.3413672 ]]

 [[ 2.9373045   2.65590706]
  [ 2.9373045   2.65590706]]

 [[ 4.36204553  3.94415611]
  [ 4.36204553  3.94415611]]

 [[ 5.75829174  5.20664019]
  [ 5.75829174  5.20664019]]

 [[ 7.12661302  6.44387459]
  [ 7.12661302  6.44387459]]

 [[ 8.46756788  7.6563643 ]
  [ 8.46756788  7.6563643 ]]

 [[ 9.78170364  8.84460421]
  [ 9.78170364  8.84460421]]

 [[11.06955669 10.00907933]
  [11.06955669 10.00907933]]

 [[12.33165268 11.15026494]
  [12.33165268 11.15026494]]

 [[13.56850674 12.26862684]
  [13.56850674 12.26862684]]

 [[14.78062373 13.3646215 ]
  [14.78062373 13.3646215 ]]]
```

```
# when the initial state is state 2
print(f'α = {ex1_finite.wealth_distribution(s0_idx=1)}')
print(f'ψ = \n{ex1_finite.continuation_wealths()}\n')
print(f'J = \n{ex1_finite.value_functionss()}')
```

```
α = [0.44981649 0.55018351]
ψ =
[[[-0.55018351  0.55018351]
  [ 0.44981649 -0.44981649]]

 [[-0.59936335  0.59936335]
  [ 0.40063665 -0.40063665]]

 [[-0.64755959  0.64755959]
  [ 0.35244041 -0.35244041]]

 [[-0.69479191  0.69479191]
  [ 0.30520809 -0.30520809]]

 [[-0.74107958  0.74107958]
```

(continues on next page)

(continued from previous page)

```
  [ 0.25892042 -0.25892042]]

 [[-0.78644149  0.78644149]
  [ 0.21355851 -0.21355851]]

 [[-0.83089617  0.83089617]
  [ 0.16910383 -0.16910383]]

 [[-0.87446176  0.87446176]
  [ 0.12553824 -0.12553824]]

 [[-0.91715603  0.91715603]
  [ 0.08284397 -0.08284397]]

 [[-0.95899642  0.95899642]
  [ 0.04100358 -0.04100358]]

 [[-1.          1.        ]
  [-0.         -0.        ]]]

J =
[[[ 1.3413672   1.48348712]
  [ 1.3413672   1.48348712]]

 [[ 2.65590706  2.9373045 ]
  [ 2.65590706  2.9373045 ]]

 [[ 3.94415611  4.36204553]
  [ 3.94415611  4.36204553]]

 [[ 5.20664019  5.75829174]
  [ 5.20664019  5.75829174]]

 [[ 6.44387459  7.12661302]
  [ 6.44387459  7.12661302]]

 [[ 7.6563643   8.46756788]
  [ 7.6563643   8.46756788]]

 [[ 8.84460421  9.78170364]
  [ 8.84460421  9.78170364]]

 [[10.00907933 11.06955669]
  [10.00907933 11.06955669]]

 [[11.15026494 12.33165268]
  [11.15026494 12.33165268]]

 [[12.26862684 13.56850674]
  [12.26862684 13.56850674]]

 [[13.3646215  14.78062373]
  [13.3646215  14.78062373]]]
```

We can check the results with finite horizon converges to the ones with infinite horizon as $T \to \infty$.

```
ex1_large = RecurCompetitive(s, P, ys, T=10000)
ex1_large.wealth_distribution(s0_idx=1)
```

```
array([0.49, 0.51])
```

```
ex1.V, ex1_large.V[-1]
```

```
(array([[[25.5, 24.5],
         [24.5, 25.5]]]),
 array([[25.5, 24.5],
        [24.5, 25.5]]))
```

```
ex1_large.continuation_wealths()
ex1.ψ, ex1_large.ψ[-1]
```

```
(array([[[-1.,  1.],
         [ 0., -0.]]]),
 array([[-1.,  1.],
        [ 0., -0.]]))
```

```
ex1_large.value_functionss()
ex1.J, ex1_large.J[-1]
```

```
(array([[[70.        , 71.41428429],
         [70.        , 71.41428429]]]),
 array([[70.        , 71.41428429],
        [70.        , 71.41428429]]))
```

CHAPTER

SIXTYONE

HETEROGENEOUS BELIEFS AND BUBBLES

Contents

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

```
!conda install -y quantecon
```

61.1 Overview

This lecture describes a version of a model of Harrison and Kreps [HK78].

The model determines the price of a dividend-yielding asset that is traded by two types of self-interested investors.

The model features

- heterogeneous beliefs
- incomplete markets
- short sales constraints, and possibly ...
- (leverage) limits on an investor's ability to borrow in order to finance purchases of a risky asset

Let's start with some standard imports:

```
import numpy as np
import quantecon as qe
import scipy.linalg as la
```

61.1.1 References

Prior to reading the following, you might like to review our lectures on

- *Markov chains*
- *Asset pricing with finite state space*

61.1.2 Bubbles

Economists differ in how they define a *bubble*.

The Harrison-Kreps model illustrates the following notion of a bubble that attracts many economists:

> *A component of an asset price can be interpreted as a bubble when all investors agree that the current price of the asset exceeds what they believe the asset's underlying dividend stream justifies.*

61.2 Structure of the Model

The model simplifies things by ignoring alterations in the distribution of wealth among investors who have hard-wired different beliefs about the fundamentals that determine asset payouts.

There is a fixed number A of shares of an asset.

Each share entitles its owner to a stream of dividends $\{d_t\}$ governed by a Markov chain defined on a state space $S \in \{0, 1\}$.

Thus, the stock is traded **ex dividend**.

The dividend obeys

$$d_t = \begin{cases} 0 & \text{if } s_t = 0 \\ 1 & \text{if } s_t = 1 \end{cases}$$

The owner of a share at the beginning of time t is entitled to the dividend paid at time t.

The owner of the share at the beginning of time t is also entitled to sell the share to another investor during time t.

Two types $h = a, b$ of investors differ only in their beliefs about a Markov transition matrix P with typical element

$$P(i, j) = \mathbb{P}\{s_{t+1} = j \mid s_t = i\}$$

Investors of type a believe the transition matrix

$$P_a = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

Investors of type b think the transition matrix is

$$P_b = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{4} & \frac{3}{4} \end{bmatrix}$$

Thus, in state 0, a type a investor is more optimistic about next period's dividend than is investor b.

But in state 1, a type a investor is more pessimistic about next period's dividend than is investor b.

The stationary (i.e., invariant) distributions of these two matrices can be calculated as follows:

```
qa = np.array([[1/2, 1/2], [2/3, 1/3]])
qb = np.array([[2/3, 1/3], [1/4, 3/4]])
mca = qe.MarkovChain(qa)
mcb = qe.MarkovChain(qb)
mca.stationary_distributions
```

```
array([[0.57142857, 0.42857143]])
```

```
mcb.stationary_distributions
```

```
array([[0.42857143, 0.57142857]])
```

The stationary distribution of P_a is approximately $\pi_a = \begin{bmatrix} .57 & .43 \end{bmatrix}$.

The stationary distribution of P_b is approximately $\pi_b = \begin{bmatrix} .43 & .57 \end{bmatrix}$.

Thus, a type a investor is more pessimistic on average.

61.2.1 Ownership Rights

An owner of the asset at the end of time t is entitled to the dividend at time $t+1$ and also has the right to sell the asset at time $t+1$.

Both types of investors are risk-neutral and both have the same fixed discount factor $\beta \in (0,1)$.

In our numerical example, we'll set $\beta = .75$, just as Harrison and Kreps did.

We'll eventually study the consequences of two alternative assumptions about the number of shares A relative to the resources that our two types of investors can invest in the stock.

1. Both types of investors have enough resources (either wealth or the capacity to borrow) so that they can purchase the entire available stock of the asset[1].
2. No single type of investor has sufficient resources to purchase the entire stock.

Case 1 is the case studied in Harrison and Kreps.

In case 2, both types of investors always hold at least some of the asset.

61.2.2 Short Sales Prohibited

No short sales are allowed.

This matters because it limits how pessimists can express their opinion.

- They **can** express themselves by selling their shares.
- They **cannot** express themsevles more loudly by artificially "manufacturing shares" – that is, they cannot borrow shares from more optimistic investors and then immediately sell them.

[1] By assuming that both types of agents always have "deep enough pockets" to purchase all of the asset, the model takes wealth dynamics off the table. The Harrison-Kreps model generates high trading volume when the state changes either from 0 to 1 or from 1 to 0.

61.2.3 Optimism and Pessimism

The above specifications of the perceived transition matrices P_a and P_b, taken directly from Harrison and Kreps, build in stochastically alternating temporary optimism and pessimism.

Remember that state 1 is the high dividend state.

- In state 0, a type a agent is more optimistic about next period's dividend than a type b agent.
- In state 1, a type b agent is more optimistic about next period's dividend.

However, the stationary distributions $\pi_a = \begin{bmatrix} .57 & .43 \end{bmatrix}$ and $\pi_b = \begin{bmatrix} .43 & .57 \end{bmatrix}$ tell us that a type B person is more optimistic about the dividend process in the long run than is a type A person.

61.2.4 Information

Investors know a price function mapping the state s_t at t into the equilibrium price $p(s_t)$ that prevails in that state.

This price function is endogenous and to be determined below.

When investors choose whether to purchase or sell the asset at t, they also know s_t.

61.3 Solving the Model

Now let's turn to solving the model.

We'll determine equilibrium prices under a particular specification of beliefs and constraints on trading selected from one of the specifications described above.

We shall compare equilibrium price functions under the following alternative assumptions about beliefs:

1. There is only one type of agent, either a or b.
2. There are two types of agents differentiated only by their beliefs. Each type of agent has sufficient resources to purchase all of the asset (Harrison and Kreps's setting).
3. There are two types of agents with different beliefs, but because of limited wealth and/or limited leverage, both types of investors hold the asset each period.

61.3.1 Summary Table

The following table gives a summary of the findings obtained in the remainder of the lecture (in an exercise you will be asked to recreate the table and also reinterpret parts of it).

The table reports implications of Harrison and Kreps's specifications of P_a, P_b, β.

s_t	0	1
p_a	1.33	1.22
p_b	1.45	1.91
p_o	1.85	2.08
p_p	1	1
$\hat{p}_a$	1.85	1.69
$\hat{p}_b$	1.69	2.08

Here

- p_a is the equilibrium price function under homogeneous beliefs P_a
- p_b is the equilibrium price function under homogeneous beliefs P_b
- p_o is the equilibrium price function under heterogeneous beliefs with optimistic marginal investors
- p_p is the equilibrium price function under heterogeneous beliefs with pessimistic marginal investors
- $\hat{p}_a$ is the amount type a investors are willing to pay for the asset
- $\hat{p}_b$ is the amount type b investors are willing to pay for the asset

We'll explain these values and how they are calculated one row at a time.

The row corresponding to p_o applies when both types of investor have enough resources to purchse the entire stock of the asset and strict short sales constraints prevail so that temporarily optimistic investors always price the asset.

The row corresponding to p_p would apply if neither type of investor has enough resources to purchase the entire stock of the asset and both types must hold the asset.

The row corresponding to p_p would also apply if both types have enough resources to buy the entire stock of the asset but short sales are also possible so that temporarily pessimistic investors price the asset.

61.3.2 Single Belief Prices

We'll start by pricing the asset under homogeneous beliefs.

(This is the case treated in *the lecture* on asset pricing with finite Markov states)

Suppose that there is only one type of investor, either of type a or b, and that this investor always "prices the asset".

Let $p_h = \begin{bmatrix} p_h(0) \\ p_h(1) \end{bmatrix}$ be the equilibrium price vector when all investors are of type h.

The price today equals the expected discounted value of tomorrow's dividend and tomorrow's price of the asset:

$$p_h(s) = \beta \left(P_h(s,0)(0 + p_h(0)) + P_h(s,1)(1 + p_h(1)) \right), \quad s = 0, 1$$

These equations imply that the equilibrium price vector is

$$\begin{bmatrix} p_h(0) \\ p_h(1) \end{bmatrix} = \beta [I - \beta P_h]^{-1} P_h \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1}$$

The first two rows of the table report $p_a(s)$ and $p_b(s)$.

Here's a function that can be used to compute these values

```
def price_single_beliefs(transition, dividend_payoff, β=.75):
    """
    Function to Solve Single Beliefs
    """
    # First compute inverse piece
    imbq_inv = la.inv(np.eye(transition.shape[0]) - β * transition)

    # Next compute prices
    prices = β * imbq_inv @ transition @ dividend_payoff

    return prices
```

Single Belief Prices as Benchmarks

These equilibrium prices under homogeneous beliefs are important benchmarks for the subsequent analysis.

- $p_h(s)$ tells what investor h thinks is the "fundamental value" of the asset.
- Here "fundamental value" means the expected discounted present value of future dividends.

We will compare these fundamental values of the asset with equilibrium values when traders have different beliefs.

61.3.3 Pricing under Heterogeneous Beliefs

There are several cases to consider.

The first is when both types of agents have sufficient wealth to purchase all of the asset themselves.

In this case, the marginal investor who prices the asset is the more optimistic type so that the equilibrium price $\bar{p}$ satisfies Harrison and Kreps's key equation:

$$\bar{p}(s) = \beta \max \left\{ P_a(s,0)\bar{p}(0) + P_a(s,1)(1+\bar{p}(1)),\ P_b(s,0)\bar{p}(0) + P_b(s,1)(1+\bar{p}(1)) \right\} \tag{2}$$

for $s = 0, 1$.

In the above equation, the max on the right side is evidently over two prospective values of next period's payout from owning the asset.

The marginal investor who prices the asset in state s is of type a if

$$P_a(s,0)\bar{p}(0) + P_a(s,1)(1+\bar{p}(1)) > P_b(s,0)\bar{p}(0) + P_b(s,1)(1+\bar{p}(1))$$

The marginal investor is of type b if

$$P_a(s,1)\bar{p}(0) + P_a(s,1)(1+\bar{p}(1)) < P_b(s,1)\bar{p}(0) + P_b(s,1)(1+\bar{p}(1))$$

Thus the marginal investor is the (temporarily) optimistic type.

Equation (2) is a functional equation that, like a Bellman equation, can be solved by

- starting with a guess for the price vector $\bar{p}$ and
- iterating to convergence on the operator that maps a guess $\bar{p}^j$ into an updated guess $\bar{p}^{j+1}$ defined by the right side of (2), namely

$$\bar{p}^{j+1}(s) = \beta \max \left\{ P_a(s,0)\bar{p}^j(0) + P_a(s,1)(1+\bar{p}^j(1)),\ P_b(s,0)\bar{p}^j(0) + P_b(s,1)(1+\bar{p}^j(1)) \right\} \tag{3}$$

for $s = 0, 1$.

The third row of the table labeled p_o reports equilibrium prices that solve the functional equation when $\beta = .75$.

Here the type that is optimistic about s_{t+1} prices the asset in state s_t.

It is instructive to compare these prices with the equilibrium prices for the homogeneous belief economies that solve under beliefs P_a and P_b reported in the rows labeled p_a and p_b, respectively.

Equilibrium prices p_o in the heterogeneous beliefs economy evidently exceed what any prospective investor regards as the fundamental value of the asset in each possible state.

Nevertheless, the economy recurrently visits a state that makes each investor want to purchase the asset for more than he believes its future dividends are worth.

The reason that an investor is willing to pay more than what he believes is warranted by fundamental value of the prospective dividend stream is he expects to have the option to sell the asset later to another investor who will value the asset more highly than he will.

- Investors of type a are willing to pay the following price for the asset

$$\hat{p}_a(s) = \begin{cases} \bar{p}(0) & \text{if } s_t = 0 \\ \beta(P_a(1,0)\bar{p}(0) + P_a(1,1)(1+\bar{p}(1))) & \text{if } s_t = 1 \end{cases}$$

- Investors of type b are willing to pay the following price for the asset

$$\hat{p}_b(s) = \begin{cases} \beta(P_b(0,0)\bar{p}(0) + P_b(0,1)(1+\bar{p}(1))) & \text{if } s_t = 0 \\ \bar{p}(1) & \text{if } s_t = 1 \end{cases}$$

Evidently, $\hat{p}_a(1) < \bar{p}(1)$ and $\hat{p}_b(0) < \bar{p}(0)$.

Investors of type a want to sell the asset in state 1 while investors of type b want to sell it in state 0.

- The asset changes hands whenever the state changes from 0 to 1 or from 1 to 0.
- The valuations $\hat{p}_a(s)$ and $\hat{p}_b(s)$ are displayed in the fourth and fifth rows of the table.
- Even the pessimistic investors who don't buy the asset think that it is worth more than they think future dividends are worth.

Here's code to solve for $\bar{p}$, $\hat{p}_a$ and $\hat{p}_b$ using the iterative method described above

```
def price_optimistic_beliefs(transitions, dividend_payoff, β=.75,
                             max_iter=50000, tol=1e-16):
    """
    Function to Solve Optimistic Beliefs
    """
    # We will guess an initial price vector of [0, 0]
    p_new = np.array([[0], [0]])
    p_old = np.array([[10.], [10.]])

    # We know this is a contraction mapping, so we can iterate to conv
    for i in range(max_iter):
        p_old = p_new
        p_new = β * np.max([q @ p_old
                            + q @ dividend_payoff for q in transitions],
                            1)

        # If we succeed in converging, break out of for loop
        if np.max(np.sqrt((p_new - p_old)**2)) < tol:
            break

    ptwiddle = β * np.min([q @ p_old
                           + q @ dividend_payoff for q in transitions],
                           1)

    phat_a = np.array([p_new[0], ptwiddle[1]])
    phat_b = np.array([ptwiddle[0], p_new[1]])

    return p_new, phat_a, phat_b
```

61.3.4 Insufficient Funds

Outcomes differ when the more optimistic type of investor has insufficient wealth — or insufficient ability to borrow enough — to hold the entire stock of the asset.

In this case, the asset price must adjust to attract pessimistic investors.

Instead of equation (2), the equilibrium price satisfies

$$\check{p}(s) = \beta \min \{P_a(s,1)\check{p}(0) + P_a(s,1)(1+\check{p}(1)),\ P_b(s,1)\check{p}(0) + P_b(s,1)(1+\check{p}(1))\} \tag{4}$$

and the marginal investor who prices the asset is always the one that values it *less* highly than does the other type.

Now the marginal investor is always the (temporarily) pessimistic type.

Notice from the sixth row of that the pessimistic price p_o is lower than the homogeneous belief prices p_a and p_b in both states.

When pessimistic investors price the asset according to (4), optimistic investors think that the asset is underpriced.

If they could, optimistic investors would willingly borrow at a one-period risk-free gross interest rate β^{-1} to purchase more of the asset.

Implicit constraints on leverage prohibit them from doing so.

When optimistic investors price the asset as in equation (2), pessimistic investors think that the asset is overpriced and would like to sell the asset short.

Constraints on short sales prevent that.

Here's code to solve for $\check{p}$ using iteration

```
def price_pessimistic_beliefs(transitions, dividend_payoff, β=.75,
                              max_iter=50000, tol=1e-16):
    """
    Function to Solve Pessimistic Beliefs
    """
    # We will guess an initial price vector of [0, 0]
    p_new = np.array([[0], [0]])
    p_old = np.array([[10.], [10.]])

    # We know this is a contraction mapping, so we can iterate to conv
    for i in range(max_iter):
        p_old = p_new
        p_new = β * np.min([q @ p_old
                            + q @ dividend_payoff for q in transitions],
                           1)

        # If we succeed in converging, break out of for loop
        if np.max(np.sqrt((p_new - p_old)**2)) < tol:
            break

    return p_new
```

61.3.5 Further Interpretation

[Sch14] interprets the Harrison-Kreps model as a model of a bubble — a situation in which an asset price exceeds what every investor thinks is merited by his or her beliefs about the value of the asset's underlying dividend stream.

Scheinkman stresses these features of the Harrison-Kreps model:

- Compared to the homogeneous beliefs setting leading to the pricing formula, high volume occurs when the Harrison-Kreps pricing formula prevails.

Type a investors sell the entire stock of the asset to type b investors every time the state switches from $s_t = 0$ to $s_t = 1$.

Type b investors sell the asset to type a investors every time the state switches from $s_t = 1$ to $s_t = 0$.

Scheinkman takes this as a strength of the model because he observes high volume during *famous bubbles*.

- If the *supply* of the asset is increased sufficiently either physically (more "houses" are built) or artificially (ways are invented to short sell "houses"), bubbles end when the supply has grown enough to outstrip optimistic investors' resources for purchasing the asset.
- If optimistic investors finance their purchases by borrowing, tightening leverage constraints can extinguish a bubble.

Scheinkman extracts insights about the effects of financial regulations on bubbles.

He emphasizes how limiting short sales and limiting leverage have opposite effects.

61.4 Exercises

61.4.1 Exercise 1

This exercise invites you to recreate the summary table using the functions we have built above.

s_t	0	1
p_a	1.33	1.22
p_b	1.45	1.91
p_o	1.85	2.08
p_p	1	1
$\hat{p}_a$	1.85	1.69
$\hat{p}_b$	1.69	2.08

You will want first to define the transition matrices and dividend payoff vector.

In addition, below we'll add an interpretation of the row corresponding to p_o by inventing two additional types of agents, one of whom is **permanently optimistic**, the other who is **permanently pessimistic**.

We construct subjective transition probability matrices for our permanently optimistic and permanently pessimistic investors as follows.

The permanently optimistic investors(i.e., the investor with the most optimistic beliefs in each state) believes the transition matrix

$$P_o = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{3}{4} \end{bmatrix}$$

The permanently pessimistic investor believes the transition matrix

$$P_p = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

We'll use these transition matrices when we present our solution of exercise 1 below.

61.5 Solutions

61.5.1 Exercise 1

First, we will obtain equilibrium price vectors with homogeneous beliefs, including when all investors are optimistic or pessimistic.

```
qa = np.array([[1/2, 1/2], [2/3, 1/3]])     # Type a transition matrix
qb = np.array([[2/3, 1/3], [1/4, 3/4]])     # Type b transition matrix
# Optimistic investor transition matrix
qopt = np.array([[1/2, 1/2], [1/4, 3/4]])
# Pessimistic investor transition matrix
qpess = np.array([[2/3, 1/3], [2/3, 1/3]])

dividendreturn = np.array([[0], [1]])

transitions = [qa, qb, qopt, qpess]
labels = ['p_a', 'p_b', 'p_optimistic', 'p_pessimistic']

for transition, label in zip(transitions, labels):
    print(label)
    print("=" * 20)
    s0, s1 = np.round(price_single_beliefs(transition, dividendreturn), 2)
    print(f"State 0: {s0}")
    print(f"State 1: {s1}")
    print("-" * 20)
```

```
p_a
====================
State 0: [1.33]
State 1: [1.22]
--------------------
p_b
====================
State 0: [1.45]
State 1: [1.91]
--------------------
p_optimistic
====================
State 0: [1.85]
State 1: [2.08]
--------------------
p_pessimistic
====================
State 0: [1.]
State 1: [1.]
--------------------
```

We will use the price_optimistic_beliefs function to find the price under heterogeneous beliefs.

```
opt_beliefs = price_optimistic_beliefs([qa, qb], dividendreturn)
labels = ['p_optimistic', 'p_hat_a', 'p_hat_b']

for p, label in zip(opt_beliefs, labels):
    print(label)
    print("=" * 20)
```

(continues on next page)

(continued from previous page)

```
    s0, s1 = np.round(p, 2)
    print(f"State 0: {s0}")
    print(f"State 1: {s1}")
    print("-" * 20)
```

```
p_optimistic
====================
State 0: [1.85]
State 1: [2.08]
--------------------
p_hat_a
====================
State 0: [1.85]
State 1: [1.69]
--------------------
p_hat_b
====================
State 0: [1.69]
State 1: [2.08]
--------------------
```

Notice that the equilibrium price with heterogeneous beliefs is equal to the price under single beliefs with **permanently optimistic** investors - this is due to the marginal investor in the heterogeneous beliefs equilibrium always being the type who is temporarily optimistic.
